Skaalautuvat arkkitehtuurit: perusta sosiaaliselle medialle

Olavi Karppi

Tampereen yliopisto
Informaatiotieteiden yksikkö
Tietojenkäsittelyoppi
Pro gradu -tutkielma
Ohjaaja: Jyrki Nummenmaa
27.5.2015

Tampereen yliopisto
Informaatiotieteiden yksikkö
Tietojenkäsittelyoppi
Olavi Karppi: Skaalautuvat arkkitehtuurit: perusta sosiaaliselle medialle
Pro gradu -tutkielma, 109 sivua
Toukokuu 2015

Tiivistelmä

Viimeisen vuosikymmenen aikana sosiaalisen median nousu on luonut uusia tapoja sekä sisällön tuottamiseen että sisällön seuraamiseen ja etsimiseen. Suositut sosiaalisen median palvelut keräävät miljoonia, jopa satoja miljoonia käyttäjiä. Sosiaalisen median myötä kuka tahansa näistä edellä mainituista käyttäjistä voi astua sisällöntuottajan rooliin. Tämä suuri määrä potentiaalisia sisällöntuottajia johtaa suuriin datamääriin, joiden hallinta johtaa haasteisiin sekä ohjelmisto- että palvelinarkkitehtuureissa.

Tämän pro gradu -tutkielman tavoitteena on tehdä katsaus siihen, miten suositut sosiaalisen median palvelut, kuten Twitter, Reddit ja Wikipedia ovat ratkaisseet nämä haasteet.

Avainsanat ja -sanonnat: big data, horisontaalinen skaalautuvuus, NoSQL, ohjelmistoarkkitehtuuri, sosiaalinen media.

Sisällys

1 Johdanto

Sosiaalinen media on lyönyt itsensä läpi 2000-luvulla. WWW-sivustojen suosituimmuutta mittaava Alexa listaa vuonna 2015 kymmenen suosituimman internetsivuston joukkoon neljä sosiaalisen median sivustoa [Alexa, 2015]. Tältä listalta löytyvät sosiaalisen median ja sosiaalisen verkostoitumisen sivusto Facebook, videoiden jakopalvelu Youtube, yhteisöllinen tietosanakirja Wikipedia sekä mikroblogipalvelu Twitter.

Historiallisesti ensimmäinen sosiaalisen verkostoitumisen palvelu oli todennäköisesti vuonna 1997 perustettu Sixdegrees-sivusto. Sixdegrees sulki ovensa vuonna 2001, ja vaikkei Sixdegrees saavuttanutkaan taloudellista menestystä, seurasi sitä pian joukko uusia sosiaalisen verkostoitumisen sekä sosiaalisen median palveluja. [Boyd & Ellison, 2007] Tällaisia palveluja olivat muun muassa Friendster, MySpace ja Facebook. 2000- ja 2010-lukujen aikana sosiaalisen median palvelujen määrä on kasvanut runsaasti. Vuonna 2015 Wikipedia listaa yli kaksisataa sosiaaliseen mediaan sekä sosiaaliseen verkostoitumiseen keskittyvää palvelua [Wikipedia, 2015b].

Sosiaalinen media terminä tarkoittaa mediaa, jossa käyttäjät ottavat aiempaa aktiivisemman roolin ja osallistuvat palvelun sisällön kuluttamisen ohella myös sen luomiseen. Perinteisillä verkkosivustoilla sisällön synnyttämisestä on usein vastannut sivuston ylläpitäjä tai omistaja. Esimerkiksi uutissivustolla uutisartikkelien kirjoittamisesta on saattanut vastata pieni joukko journalisteja. Uutissivuston käyttäjien rooliksi on jäänyt ainoastaan uutisten lukeminen, eli sisällön kuluttaminen. Sosiaalisen median erottaa perinteisestä mediasta se, että *sosiaalisessa mediassa suuri osa käyttäjistä toimii sekä sisällöntuottajina että sisällönkuluttajina*. Sosiaalisen median palvelussa päivittäistä sisältöä saattaa tuottaa kymmenen tai sata miljoonaa ihmistä. Sosiaalinen media tarjoaakin ennennäkemättömät mahdollisuudet ilmaista omat mielipiteensä ja ajatuksensa. Vastapainona kasvaneelle sisällön määrälle, sosiaalisen median palvelut tarjoavat myös uusia tapoja löytää sisältöä, esimerkiksi mahdollistamalla ainoastaan valikoitujen käyttäjien luoman sisällön seuraamisen.

Sosiaalinen media ja sosiaalinen verkostoituminen esiintyvät usein sosiaalisissa palveluissa rinnakkain. Siinä missä sosiaalisen median palvelun voidaan luonnehtia sallivan käyttäjien luoda sisältöä palveluunsa, voidaan sosiaalista verkostoitumista sallivaa palvelua luonnehtia siten, että se sallii käyttäjien itsensä muodostamat sosiaaliset verkostot.

Facebook, yksi suosituimmista sosiaalisen median palveluista, keräsi vuonna

2013 yli 1,2 miljardia kuukausittaista käyttäjää [Kiss, 2014]. Monet vähemmänkin suositut palvelut keräävät kymmeniä tai satoja miljoonia kuukausittaisia käyttäjiä. Mikroblogipalvelu Twitter arvioi vuonna 2013 kuukausittaiseksi käyttäjämääräkseen yli 200 miljoonaa käyttäjää [Sec, 2013a]. Linkkien ja uutisten jakamiseen käytetty, internetin etusivuksi itseään kutsuva, Reddit arvioi vuoden 2013 kuukausittaiseksi käyttäjämääräkseen 60 miljoonaa käyttäjää [Martin, 2013]. Arvioiden mukaan vuonna 2013 jopa 73 % aikuisista internetin käyttäjistä käytti vähintään yhtä sosiaalisen median palvelua [Duggan & Smith, 2013b]. Sosiaalisen median käyttäjät käyttävät palveluja myös usein hyvin aktiivisesti. Esimerkiksi Facebookin käyttäjistä jopa 63 % käytti sivustoa päivittäin [Duggan & Smith, 2013a]. Tämä tarkoittaisi reilua kuuttasataa miljoonaa päivittäistä käyttäjää. Lisäksi Facebookin arvioitiin tallentavan dataa päivittäin yli 500 teratavua vuonna 2012 [Constine, 2012].

Sosiaalisen median keräämät suuret käyttäjämäärät sekä käyttäjien aiemmasta eroavat käyttäytymismallit ovat johtaneet uudenlaisiin suorituskyky- ja tehovaatimuksiin sekä ohjelmisto- että palvelinarkkitehtuureissa. Esimerkiksi Facebookin on arvioitu hyödyntävän toistasataatuhatta palvelinta palvellakseen käyttäjiään [Higginbotham & Kern, 2013, Gruener, 2012]. Tämä eroaa merkittävästi esimerkiksi perinteisistä yritysjärjestelmistä, jotka ovat useimmiten koostuneet vain muutamasta tai maksimissaan muutamasta kymmenestä palvelimesta.

Tämän pro gradu -tutkielman tavoitteena on tutkia, minkälaisia ohjelmisto- sekä palvelinarkkitehtuuriratkaisuja suositut sosiaalisen median palvelut ovat päätyneet käyttämään hyväkseen. Tavoitteena on tutkia onko tutkielmassa läpikäytävien palvelujen taustalta löydettävissä yleisesti käytettyjä arkkitehtuurillisia suunnittelumalleja tai piirteitä, jotka takaavat niille niiden kyvyn skaalautua.

2 Termien määrittely

Voidaksemme paremmin ymmärtää, minkälaisia haasteita sosiaalisen median palveluiden ohjelmistoarkkitehtuureissa on tarve ratkaista, lähdetään liikkeelle tutkimalla, mitä termit sosiaalinen media, arkkitehtuuri ja skaalautuvuus tarkoittavat.

2.1 Sosiaalisen median piirteet ja kuvaus

Sosiaalisen median palveluksi voidaan kuvailla palvelua, jossa palvelun käyttäjät ovat päävastuussa sekä palvelun sisällön tuottamisesta että myös sen kuluttamisesta.

Monet sosiaalisen median palvelut muistuttavat tietyiltä piirteiltään toisiaan. Yhteisiä piirteitä ovat esimerkiksi palveluun rekisteröityneet käyttäjät, käyttäjien palveluun tuoma sisältö sekä usein tämän sisällön ympärille muodostuvat keskustelut.

Vaikka palveluista löytyy yhtenäisiä piirteitä, ei tämä kuitenkaan tarkoita sitä, että palvelut soveltuisivat vain yhteen käyttötarkoitukseen. Päinvastoin sosiaalisen median palvelut soveltuvat useisiin erilaisiin käyttötarkoituksiin. Taulukkoon 2.1 on listattu joitain vuonna 2014 toiminnassa olleita suosittuja sosiaalisen median ja sosiaalisen verkostoitumisen palveluja. Taulukko 2.1 ei kata kaikkia sosiaalisen median palveluja, mutta se osoittaa kuitenkin sen, että sosiaalisen median palveluja käytetään hyvin vaihtelevasti erilaisiin käyttötarkoituksiin. Kattavampi lista löytyy esimerkiksi Wikipediasta, joka listaa yli kaksisataa sosiaaliseen mediaan ja sosiaaliseen verkostoitumiseen keskittyvää palvelua [Wikipedia, 2015b].

Sosiaalisen median palvelujen yhteisiä piirteitä on kuvailtu eri henkilöiden toimesta. Tutustutaan seuraavaksi siihen, miten Boyd ja Ellison [2007] sekä Kietzmann *et al.* [2011] luonnehtivat näitä piirteitä. Boyd ja Ellison [2007] määrittelevät sosiaalisen verkostoitumisen ja median palveluiksi palvelut, jotka täyttävät seuraavat ehdot:

1. Palvelu mahdollistaa julkisen tai osittain julkisen profiilin luomisen ja tallentamisen.

2. Palvelu mahdollistaa käyttäjälle tavan kertoa, keiden muiden käyttäjien kanssa hänellä on kontakti.

3. Palvelu mahdollistaa omien kontaktien listaamisen sekä heidän profiiliensa läpikäynnin.

Palvelun WWW-osoite	**Kuvaus**
https://www.facebook.com/	Facebook on yhteisöpalvelu.
https://twitter.com/	Twitter on mikroblogipalvelu, joka sallii maksimissaan 140 merkin mittaisten viestien julkaisemisen.
http://www.reddit.com/	Reddit on palvelu uutisten, linkkien ja keskusteluiden jakamiseen.
http://www.last.fm/	Last.fm tallentaa tiedon siitä, mitä musiikkikappaleita kukin sen käyttäjä kuuntelee ja sallii tämän tiedon jakamisen muille käyttäjille.
https://www.fitocracy.com/	Fitocracy on palvelu liikuntasuoritusten seurantaan ja jakamiseen.
https://www.flickr.com/	Flickr on palvelu kuvien jakamiseen.
https://www.youtube.com/	Youtube on palvelu videoiden jakamiseen.
https://www.linkedin.com/	LinkedIn on palvelu ammatillisten profiilien jakamiseen ja ammatilliseen verkostoitumiseen.
http://www.wikipedia.org/	Wikipedia on yhteisöllisesti kirjoitettu tietosanakirja.
https://www.tumblr.com/	Tumblr on palvelu blogien kirjoittamiseen.
https://www.couchsurfing.com/	Couchsurfing on palvelu ilmaisten tai halpojen yksityisten ihmisten tarjoamien majoituspaikkojen tarjoamiseen.

Taulukko 2.1: Suosittuja sosiaalisen verkostoitumisen ja sosiaalisen median palveluja.

Kietzmann *et al.* [2011] pyrkivät luonnehtimaan sosiaalisen median palveluja hieman tarkemmalla tasolla. Kietzmann *et al.* [2011] kuvailevat sosiaalisen median palveluja käyttämällä seitsemää eri piirrettä:

- identiteetti (identity)
- läsnäolo (presence)
- jakaminen (sharing)

- relaatiot (relationships)
- keskustelut (conversations)
- maine (reputation)
- ryhmät (groups).

Identiteetillä tarkoitetaan käyttäjän palveluun jakamiaan henkilökohtaisia tietoja, kuten nimeä, ikää, sukupuolta ja ammattia. Identiteettiin voi liittyä myös muita tietoja. Esimerkiksi LinkedInissä käyttäjä saattaa kertoa tiedot koulu- ja urataustastaan. Facebookissa omaan identiteettiin saattaa liittyä tiedot siitä, mistä asioista on tykännyt.

Läsnäolo kertoo sen, onko käyttäjä paikalla kyseisessä palvelussa vai ei. Läsnäolotieto saattaa olla oleellinen esimerkiksi deittailuun tai keskusteluun keskittyvissä palveluissa. Joissain muissa palveluissa, kuten esimerkiksi LinkedInissä, tämän kaltaiselle tiedolle on vähemmän käyttöä.

Jakaminen viittaa jonkin objektin jakamiseen sosiaalisen median palvelussa. Tällainen objekti voi olla esimerkiksi video Youtubessa, twiitti Twitterissä tai linkki verkkosivulle Redditissä.

Relaatiot viittaavat käyttäjien toisiinsa muodostamiin suhteisiin. Esimerkiksi Twitterin sallima seuraaja-seurattu-suhde on relaatio. Vastaavasti relaatioita ovat linkittäytyminen toiseen käyttäjään LinkedInissä tai kaverilinkitys Facebookissa.

Keskustelut viittaavat käyttäjien keskenään käymiin keskusteluihin. Usein nämä keskittyvät palvelussa jaetun sosiaalisen objektin ympärille. Esimerkiksi Redditissä jaetun linkin ympärille syntyy usein pitkä viestiketju, johon käyttäjät kommentoivat linkin sisältöä sekä edelleen kommentoivat muiden käyttäjien aiemmin lisäämiä kommentteja.

Maine viittaa maineeseen, joka käyttäjälle on sosiaalisen median palvelussa muodostunut. Mainetta kuvaavat muun muassa Redditissä karmapisteet, Facebookissa tykkäysten määrät ja Twitterissä seuraajien lukumäärä.

Ryhmillä viitataan käyttäjien keskenään muodostamiin ryhmiin. Ryhmien sisällä voi olla mahdollista käydä esimerkiksi keskusteluja, jotka näkyvät vain ryhmän jäsenille [Kietzmann *et al.*, 2011]. Ryhmän perustaminen saattaa mahdollistaa myös esimerkiksi leikkimielisten kilpailujen järjestämisen. Näin voi tehdä esimerkiksi urheilusuorituksia seuraavassa Fitocracyssä.

Kietzmann *et al.* [2011] toteavat, ettei kaikista sosiaalisen median palveluista välttämättä löydy jokaista edellä mainittua piirrettä. Esimerkiksi läsnäolotieto ei

välttämättä ole mielenkiintoinen tieto kaikissa palveluissa ja voi olla siten jätetty pois.

2.1.1 Sosiaaliset objektit

Monia sosiaalisen median palveluja yhdistää jonkin *sosiaalisen objektin* (object of sociality) jakaminen, jonka ympärille varsinaiset sosiaaliset piirteet, kuten keskustelut, muodostuvat [Kietzmann *et al.*, 2011]. Redditissä jaetaan linkkejä, uutisia ja keskusteluja, Fitocracyssä jaetaan tietoja liikuntasuorituksista, Youtubessa jaetaan videoita, Flickr:ssä kuvia ja Tumblr:ssä blogeja. Muut palvelun käyttäjät pääsevät (usein eri tavoin rajoitetusti) näkemään muiden jakamia objekteja, ja usein muilla käyttäjillä on mahdollisuus ilmaista mielipiteensä näistä objekteista. Mielipiteen ilmaisu voi tapahtua kommentoimalla kyseistä objektia, kuten on mahdollista Redditissä ja Youtubessa. Muita mahdollisuuksia ovat muun muassa objektin pisteyttäminen sen mukaan, kuinka hyvä tai mielenkiintoinen se oli. Tämä onnistuu esimerkiksi Facebookissa tykkäyksen antamisella ja Redditissä plus- ja miinuspisteiden antamisella. Ohjelmistoarkkitehtuurin näkökulmasta sosiaaliset objektit sekä niihin liittyvät kommentit, tykkäämiset ja muu metadata ovat usein sosiaalisen palvelun palvelimille suurimman kuorman aiheuttava osa. Esimerkiksi Facebookin arvioitiin vuonna 2012 tallentaneen päivittäin 2,7 miljardia tykkäystä [Constine, 2012].

2.1.2 Sosiaalinen verkosto

Monet sosiaalisen median palvelut käyttävät termiä sosiaalinen verkosto (social graph) kuvaamaan käyttäjien välisiä relaatioita [Pointer, 2010, Constine, 2013]. Sosiaalinen verkosto muodostuu relaatioista, joita käyttäjät muodostavat toisiinsa. Esimerkiksi Twitterissä käyttäjä voi seurata toista käyttäjää. Relaatiot voidaan tällaisessa tapauksessa esittää suunnattuna graafina.

Graafin solmusta lähtevien suunnattujen kaarien lukumäärä (ulkoaste) ja solmuun tulevien suunnattujen kaarien lukumäärä (sisäaste) voi vaihdella merkittävästi palvelusta, käyttäjästä ja relaation tyypistä riippuen. Esimerkiksi Twitterissä suosittua henkilöä saattaa seurata useampi kymmenen miljoonaa käyttäjää [Twitaholic, 2015]. Toisaalta normaalia käyttäjää seuraa useimmiten maksimissaan muutama sata muuta käyttäjää. Twitterin seuraaja-seurattu-relaatiossa graafin solmun sisäasteen arvo voi siis vaihdella nollan ja useamman kymmenen miljoonan välillä. Jotkin palvelut myös rajoittavat relaatioiden määrää. Esimerkiksi Facebookissa käyttäjällä on mahdollista olla maksimissaan 5000 ystävää

[Facebook, 2015b].

2.2 Ohjelmistoarkkitehtuuri

Tarkastellaan seuraavaksi sitä, mitä ohjelmistoarkkitehtuuri tarkoittaa ja minkälaisia käsitteitä sen yhteydessä käytetään.

Ohjelmistoarkkitehtuurin tarkoitus voidaan määritellä eri tavoin. Maier ja Rechtin [2000, 29] luonnehtivat arkkitehtuurin järjestelmän kuvaukseksi. Heidän mukaansa järjestelmä on edelleen kokoelma asioita, jotka yhdessä tuottavat tuloksia, joita osat yksinään eivät ole kykeneviä tuottamaan.

Maier ja Rechtin [2000, 31] määrittelevät edelleen onnistuneelle järjestelmälle kaksi yleispiirrettä: järjestelmän tulisi suorittaa määritellyn tehtävänsä hyväksyttävillä kustannuksilla, ja sen tulisi kyetä tekemään tämä hyväksyttävän kestoisen ajan ajan. Järjestelmän käyttämän ohjelmistoarkkitehtuurin tulisi tukea näiden tavoitteiden saavuttamista.

Toisen kuvauksen tarjoaa Fowler [2002, 11], joka toteaa arkkitehtuurin olevan korkean tason kuvaus järjestelmän osista, tai kuvaus osista ja päätöksistä, joita on hankala muuttaa.

Ohjelmisto- ja järjestelmäarkkitehtuureille on myös olemassa ISO/IEC/IEEE-standardi 42010:2011. Standardin mukaan arkkitehtuuri koostuu järjestelmän perustavanlaatuisista piirteistä, sisältäen järjestelmän elementit sekä relaatiot ja toteuttaen järjestelmän suunnittelua sekä sen evoluutiota. ISO/IEC/IEEE-standardi 42010:2011 määrittelee arkkitehtuurikuvauksen kuvaukseksi näistä piirteistä. [ISO/IEC/IEEE, 2011]

Arkkitehtuurikuvaus katsoo arkkitehtuuria eri *näkökulmista* (viewpoint) *näkymiä* (view) käyttäen. Esimerkiksi Kruchtenin hyvin tunnettu 4+1 malli [Kruchten, 1995] kategorisoi näkökulmat viiteen eri kategoriaan. Näkökulma voi vaatia järjestelmän kuvaamista esimerkiksi lähdekoodin rakenteen, luokkahierarkian, tietoturvan, suorituskyvyn, käytettävyyden tai vaikkapa skaalautuvuuden näkökulmasta. Näkökulma määrittelee, mitä tulisi kuvata ja mitä piirteitä kuvauksella tulisi olla. Näkymän tavoitteena on esittää nämä asiat kyseessä olevan järjestelmän arkkitehtuurin näkökulmasta.

2.3 Skaalautuvuus

Tehdään vielä katsaus siihen, mitä skaalautuvuudella tarkoitetaan. Yksinkertainen määritelmä löytyy esimerkiksi Dictionary.comista [Dictionary, 2015], jossa

skaalautuvuus määritellään kyvyksi sopeutua kasvavaan kysyntään. Vastaavan yksinkertaisen määritelmän tarjoaa Edberg [2013b], joka yksinkertaistaa skaalautuvan arkkitehtuurin tavoitteet siihen, että minkä tahansa käytössä olevan resurssin määrää pitää aina pystyä kasvattamaan, ja tämän tulee johtaa suorituskyvyn kasvuun.

Yksityiskohtaisempia määritelmiä tarjoavat esimerkiksi Bondi [2000] ja Fowler [2002]. Bondi [2000] määrittelee skaalautuvuuden siten, että järjestelmällä tulee olla kyky käsitellä tai prosessoida kasvavaa määrää elementtejä tai objekteja ilman, että se ajautuu tilanteeseen, jossa se ei enää kykene toimimaan. Lisäksi järjestelmän pitäisi olla tarpeen vaatiessa sellainen, että sen prosessointikykyä on mahdollista helposti kasvattaa. Bondi kuvailee ei-skaalautuvan järjestelmän sellaiseksi, jossa prosessointikykyä ei ole mahdollista kasvattaa joko ollenkaan tai ilman merkittäviä lisäkustannuksia.

Fowler [2002, 17] määrittelee skaalautuvuuden sen määreeksi, miten resurssien lisääminen vaikuttaa suorituskykyyn. Skaalautuvaksi järjestelmäksi Fowler määrittelee järjestelmän, joka sallii resurssien lisäämisen ja samassa suhteessa kasvavan kyvyn suorittaa niitä tehtäviä, joita järjestelmän tavoitteena on suorittaa.

Järjestelmän skaalautuvuus voidaan luokitella kahteen kategoriaan: horisontaaliseen sekä vertikaaliseen skaalautuvuuteen. Vertikaalinen skaalautuvuus (scale up) tarkoittaa yksittäisen resurssin tehojen kasvattamista. Esimerkiksi palvelimen prosessorin vaihtamista nopeampaan, muistin lisäämistä palvelimelle tai vaikkapa palvelimen levyjärjestelmän päivittämistä tehokkaampaan. Horisontaalinen skaalautuvuus (scale out) tarkoittaa kykyä lisätä palvelimien lukumäärää ja näiden lisättyjen palvelinten kykyä ottaa vastuulleen oma osansa palveluun kohdistuvasta kuormituksesta.

Bondi [2000] luokittelee skaalautuvan järjestelmän piirteet edelleen neljään eri kategoriaan. Hän nimeää ja kuvaa kategoriat seuraavasti:

- *Kuormitusskaalautuvuus (load scalability)*: järjestelmä on kuorman suhteen skaalautuva, jos se toimii ilman tarpeettomia viiveitä ja resurssejaan tuhlaamatta kevyen, keskitasoisen sekä raskaan kuormituksen alaisena.

- *Tilaskaalautuvuus (space scalability)*: järjestelmä on tilan suhteen skaalautuva, mikäli sen muistivaatimukset eivät kasva kohtuuttomasti, kun järjestelmän käsittelemien elementtien tai objektien määrä kasvaa.

- *Tila-aikaskaalautuvuus (space-time scalability)*: järjestelmä on tilan ja ajan

suhteen skaalautuva, mikäli sen vaatimat tila- ja aikavaatimukset eivät kasva kohtuuttomasti, vaikka käsiteltävien elementtien tai objektien määrä kasvaisi *kertaluokkaa* suuremmaksi.

- *Rakenteellinen skaalautuvuus (structural scalability)* : järjestelmä on rakenteellisesti skaalautuva, mikäli sen toteutus tai sen käyttämät standardit eivät estä käsiteltävien elementtien tai objektien määrän kasvattamista. Mikäli esimerkiksi objektit yksilöidään tunnisteella, jonka koko on 32 bittiä, loppuu käytettävissä olevien tunnisteiden määrä, kun objektien määrä ylittää 2^{32} kappaleen määrän. Tämä voi olla hyväksyttävää, jos järjestelmään tallennettavien objektien määrän ei oleteta kasvavan yli tämän määrän järjestelmän käyttöaikana.

Edellä kuvatut määritelmät keskittyvät pääosin skaalautuvuuteen järjestelmän prosessointikyvyn *kasvattamisen* näkökulmasta. Skaalautuvuutta voidaan kuitenkin katsoa myös toisesta näkökulmasta. Järjestelmän pitäisi olla myös kykenevä sopeuttamaan prosessointikykyään pienemmäksi, mikäli sille ei ole hetkellisesti tarvetta. Prosessointikykyä pienemmäksi sopeuttamalla voidaan esimerkiksi säästää palvelinten käyttökustannuksissa.

Järjestelmän kyvystä skaalautua tarpeen mukaan joko pienemmäksi tai suuremmaksi käytetään termiä elastinen skaalautuvuus [Hewitt, 2010, 16]. Elastinen skaalautuvuus on mahdollista muun muassa pilviympäristöissä (cloud environments), joissa palvelinresursseja on mahdollista ottaa käyttöön sekä poistaa käytöstä pienillä viiveillä.

Elastisuuden ohella pilviympäristöt voivat tarjota kyvyn muuttaa järjestelmän kokoa automaattisesti esimerkiksi tiettyinä kellonaikoina tai järjestelmään kohdistuvan kuorman mukaan. Esimerkiksi Amazon Web Services tarjoaa kyseisen palvelun Elastic Compute Cloud -palvelunsa ohella nimellä *auto scaling*. [Amazon, 2015c]

3 Esimerkkitapaukset

Tarkempaan tarkasteluun on valittu kolme sosiaalisen median palvelua. Kriteereinä valintaa tehtäessä on käytetty seuraavia ehtoja:

- Palvelun tulee olla riittävän suosittu, jotta sen käyttäjämäärät ja siten palvelun koko ovat mielenkiintoisia.
- Palvelun arkkitehtuurin tulee olla riittävän julkinen, jotta sen kuvaaminen ja tutkiminen on mahdollista.
- Kirjoittajalla tulee olla kokemusta palvelun käyttämisestä.

Edellä mainittujen kriteerien pohjalta tarkasteltaviksi palveluiksi on valittu *Twitter*, *Reddit* sekä *Wikipedia*. Tässä luvussa käydään edellä mainittujen sosiaalisen median palvelujen toiminnalliset piirteet sekä tutkielman aiheen kannalta mielenkiintoiset arkkitehtuurilliset piirteet läpi niiltä osin, kuin niistä löytyy julkista tietoa.

Edellä mainittujen palvelujen osalta pyritään kuvaamaan seuraavat piirteet:

- *Yleinen kuvaus*: kuvaus palvelun omaamista toiminnallisista piirteistä.
- *Laadulliset vaatimukset*: lyhyt kuvaus tutkielman kannalta oleellisista laadullisista vaatimuksista.
- *Arkkitehtuuri*: kuvaus palvelun arkkitehtuurista tietovuon, tietosäilöjen ja skaalautuvuuden näkökulmista.

Lisäksi tutkielmassa sivutaan pintapuolisemmin joitain tekniikoita, joita on käytetty *Facebookissa* ja *Instagrammissa*.

Palveluiden arkkitehtuurien kuvaukset pohjautuvat internetissä vapaasti julkaistuihin dokumentaatioihin sekä esitelmiin. Monet tutkielmassa käytetyt lähteet ovat muutaman vuoden ikäisiä, joten on oletettavaa, että tiedot ovat jo osin vanhentuneita. Uudempia lähteitä ei kaikkien tietojen osalta ole saatavilla. Lisäksi on selvää, että arkkitehtuureista on mahdollista muodostaa vain hyvin korkean tason kuvaus. Lisäksi tutkielman laajuuden puitteissa ei ole mahdollista porautua kovin syvälle eri järjestelmien yksityiskohtiin, ja toisaalta tämä ei kaikissa tapauksissa olisi mahdollistakaan julkisen tiedon puutteen vuoksi.

3.1 Twitter

3.1.1 Yleinen kuvaus

Twitter kutsuu itseään mikroblogipalveluksi. Twitterissä on mahdollista julkaista maksimissaan 140 merkin mittaisia tekstimuotoisia viestejä. Viesteistä käytetään yleisesti termiä *twiitti* (tweet) ja viestien julkaisemisesta termiä *twiittaaminen* (tweeting). Viestit ovat oletuksena julkisia, eli kaikki muut käyttäjät pääsevät lukemaan niitä.

Viestit voivat sisältää tekstisisällön lisäksi *aihetunnisteita* (hashtag). Aihetunnisteet ovat avainsanoja, jotka alkavat #-merkillä. #-merkkiä seuraa jokin vapaamuotoinen teksti, jolla ilmaistaan se, minkälaiseen aiheeseen viesti liittyy. Aihetunnisteiden ohella viestit voivat sisältää *mainintoja* (mention) toisiin käyttäjiin. Viestissä on mahdollista mainita toinen käyttäjä lisäämällä viestiin @-merkki ja heti sen perään toisen käyttäjän käyttäjätunnus.

Julkaistut viestit löytyvät Twitteristä *aikajanojen* (timeline) avulla. Twitter käyttää kyseistä termiä julkaistujen viestien käänteiseen aikajärjestykseen lajitellusta osajoukosta.

Käyttäjän kotisivulla näkyy *kotisivun aikajana* (home timeline). Toista käyttäjää *seuraamalla* (following) toisen käyttäjän viestit päivittyvät oman kotisivun aikajanaan sitä mukaa, kun seuratut käyttäjät julkaisevat viestejä.

Twitterissä on mahdollista seurata useita käyttäjiä. Seuraamissuhde voi olla yhdensuuntainen. Seuratun käyttäjän ei tarvitse hyväksyä seuraamista eikä hänen ole pakko seurata hänen *seuraajiaan* (followers). Suosituimmilla käyttäjillä saattaa olla Twitterissä jopa kymmeniä miljoonia seuraajia [Twitaholic, 2015].

Kotisivun aikajanan lisäksi käyttäjän on mahdollista luoda *listoja* (list), joihin hän lisää itse valitsemansa joukon seurattuja käyttäjiä. Käyttäjä voi näin luoda erillisiä tiettyihin aihealueisiin keskittyneitä aikajanoja.

Kotisivun aikajanan lisäksi Twitteristä löytyy myös muutama muu aikajana. *Käyttäjän aikajana* (user timeline) sisältää käänteisessä aikajärjestyksessä kaikki käyttäjän itse lähettämät viestit. Lisäksi Twitteristä löytyy hakutoiminto, joka näyttää hakutulokset *haun aikajanassa* (search timeline). Haun aikajana sisältää viestit, jotka vastaavat käyttäjän antamia hakuehtoja.

Tietyt viestin alkuun lisättävät lyhenteet tulkitaan Twitterissä tietyllä tapaa. Viesti on mahdollista *uudelleentwiitata* (**r**e-**t**weet) vastaamalla (reply) toisen käyttäjän viestiin kopioimalla toisen käyttäjän aiemmin lähettämä viesti ja lisäämällä viestin alkuun lyhenne RT ja maininta (@) toisen käyttäjän käyttäjä-

tunnuksesta.

Twitter mahdollistaa myös *yksityisten viestien* (**D**irect **M**essage) lähettämisen omille seuraajille. Yksityisen viestin lähettäminen tapahtuu vastaavasti kuin uudelleentwiittaaminen, mutta RT-lyhenteen sijaan käytetään lyhennettä DM.

Toinen tapa päästä käsiksi Twitterin dataan ovat Push-pohjaiset *www-sovelluspalvelut* (web service), joissa dataa ja päivityksiä välitetään käyttäjälle tai toiselle sovellukselle sitä mukaa, kun uutta dataa tai päivityksiä Twitteriin muodostuu. Esimerkiksi Twitterin mobiiliasiakasohjelmat ottavat vastaan Twitterin niille välittämiä päivityksiä.

Twitter tarjoaa kehittäjien käyttöön *Streaming API* -nimisen palvelun [Twitter, 2015c]. Palvelu mahdollistaa HTTP-yhteyden avaamisen Twitterin palvelimelle sekä suodattimien määrittämisen, joiden perusteella vastaanotettavat viestit valikoidaan. Yhteyden ottamisen jälkeen Twitter välittää suodattimiin osuvia viestejä ja päivityksiä asiakasohjelmistolle sitä mukaa, kun viestejä tai päivityksiä Twitteriin muodostuu.

Lisäksi Twitter tarjoaa *Firehose*-nimisen palvelun, joka välittää viestejä *kaikista* Twitterin julkisista tapahtumista kolmansille osapuolille. Streaming API rajoittaa vastaanotettavien viestien määrää sallimalla maksimissaan 5000 käyttäjätunnisteen tai maksimissaan neljänsadan avainsanan määrittämisen [Twitter, 2015b]. Firehose ei vastaavia rajoitteita aseta, vaan se välittää vastaanottavalle taholle kaikki Twitterissä tapahtuvat julkiset tapahtumat. Firehosen avulla kolmannet osapuolet voivat vastaanottaa viestejä, jotka ilmaisevat esimerkiksi uusia Twitterissä julkaistuja viestejä, muutoksia käyttäjien profiileihin sekä muun muassa muutoksia seuraaja-seurattu-suhteisiin. Firehosen käyttö on sallittu vain valikoiduille tahoille. [Twitter, 2015a]

3.1.2 Laadulliset vaatimukset

Twitter kuvailee tavoitteekseen halun olla maailman suurin viestiväylä. Twitterin tavoitteisiin kuuluu reaaliaikaisuus. Viestin lähettämisen jälkeen kyseinen viesti pitäisi näkyä maksimissaan muutaman minuutin, ja mielellään *paljon* pienemmän ajan sisällä kaikille käyttäjille, jotka seuraavat viestin lähettänyttä käyttäjää. [Krikorian, 2012b]

Twitter kertoi vuonna 2013 aktiivisten käyttäjien määräkseen noin 150 miljoonaa käyttäjää. [Krikorian, 2012b] Vuoden 2013 loppupuolella Twitterin kautta oli koko Twitterin historian aikana julkaistu kaikkineen yli 300 miljardia twiittiä [Sec, 2013b].

Twitterissä tapahtuu merkittävästi enemmän luku- kuin kirjoitusoperaatioita. Vuonna 2013 Twitter käsitteli noin 300000 lukuoperaatiota sekunnissa ja noin 5000–7000 kirjoitusoperaatiota sekunnissa. [Krikorian, 2012b]

Vuoden 2014 toisella neljänneksellä Twitterin aikajanat keräsivät 173 miljardia katselukertaa keskimääräisen kuukausittaisen määrän ollessa tällöin reilut 43 miljardia katselukertaa [Twitterinc, 2014].

Vuonna 2013 Twitterissä julkaistiin keskimäärin 5700 uutta viestiä per sekunti. Viestien määrät saattavat kuitenkin hetkittäin kasvaa merkittävästi, kuten tapahtui esimerkiksi 3.8.2013, kun Japanin televisiossa esitettiin *Laputa: linna taivaalla* -animaatioelokuva. Tällöin julkaistujen viestien määrä nousi hetkellisesti 143199 viestiin per sekunti. [Krikorian, 2013]

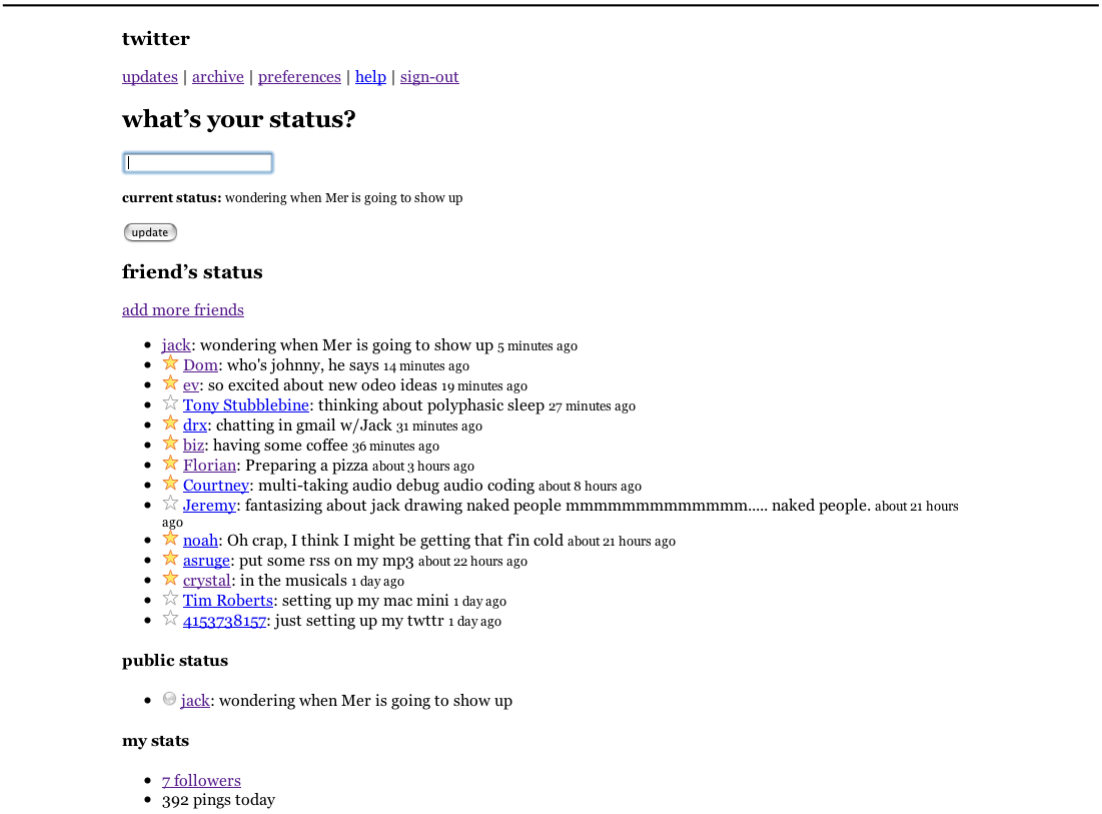
Twitterin hakupalvelu vastasi vuonna 2012 kahteen miljardiin päivittäiseen kyselyyn. Yhden haun prosessointi kesti keskimäärin 50 millisekuntia, ja keskimäärin uudet viestit olivat hakupalvelun löydettävissä 10 sekunnin kuluessa niiden luonnista. [Busch *et al.*, 2012]

Viestien julkaisemisen reaaliaikaisuusvaatimus on haasteellinen erityisesti paljon seurattujen käyttäjätilien kohdalla. Twitter päivittää kaikkien seuraajien aikajanat aina heti viestin julkaisemisen jälkeen. [Krikorian, 2012b] Sadalla suosituimmalla Twitter-käyttäjätilillä on yhdeksästä miljoonasta aina yli viiteenkymmeneen miljoonaan seuraajaa. Twitterin seuratuimmalla käyttäjällä, Kate Perryllä, oli vuonna 2015 lähemmäs 70 miljoonaa seuraajaa. [Twitaholic, 2015] Kun Kate Perry julkaisee Twitterissä viestin, pitäisi tämän viestin näkyä lähes reaaliaikaisesti 70 miljoonaan seuraajan aikajanassa. Mikäli viestit eivät päivity seuraajille riittävän nopeasti, saattaa lopputuloksena olla rikkinäiseltä vaikuttava viestinvaihto. Esimerkiksi Kate Perry julkaisee viestin ja käyttäjä A näkee viestin ennen käyttäjää B. Tällaisessa tilanteessa käyttäjä A saattaa vastata viestiin ja sopivassa tilanteessa käyttäjä B saattaa nähdä A:n vastauksen ennen Kate Perryn julkaisemaa viestiä [Krikorian, 2012b].

3.1.3 Arkkitehtuuri

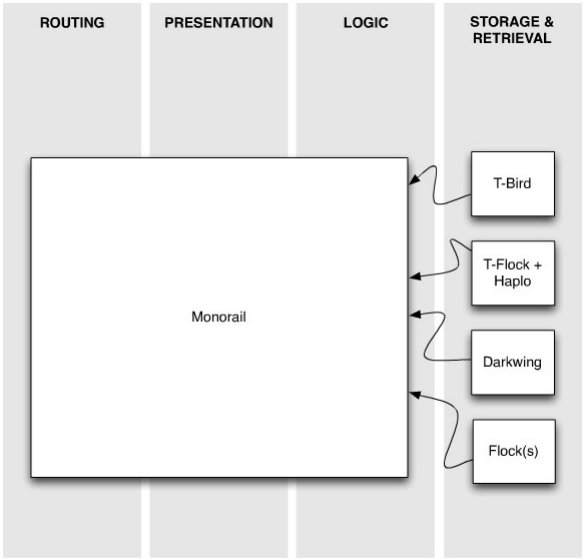
3.1.3.1 Historia

Twitter sai alkunsa vuonna 2006. Ensimmäinen versio Twitteristä näytti varsin karulta, kuten voidaan nähdä Twitterin perustajan Jack Dorseyn [2006] julkaisemasta kuvasta 3.1. Twitterin alkutaipaleella käyttäjämäärät olivat vielä pieniä ja palvelun arkkitehtuuri oli sitä myöten nykyiseen verrattuna yksinkertainen.



Kuva 3.1 Twitter-palvelun ensimmäinen versio. Kuvan on julkaissut Twitterin perustaja Jack Dorsey [2006].

Krikorian [2012a] kuvailee Twitterin alkuperäistä arkkitehtuuria neljään kerrokseen jaetuksi yhtenäiseksi sovellukseksi, jonka tavoitteena oli toteuttaa kaikki Twitterin kaipaamat toiminnot. Twitter oli tässä vaiheessa toteutettu Ruby on Railssin pohjalle ja Twitter kutsui arkkitehtuuriaan nimellä *Monorail*. Krikorianin kuvaus arkkitehtuurista näkyy kuvassa 3.2.



Kuva 3.2 Twitter-palvelun arkkitehtuuri alkuaikoina. Kuvan on julkaissut Krikorian [2012a].

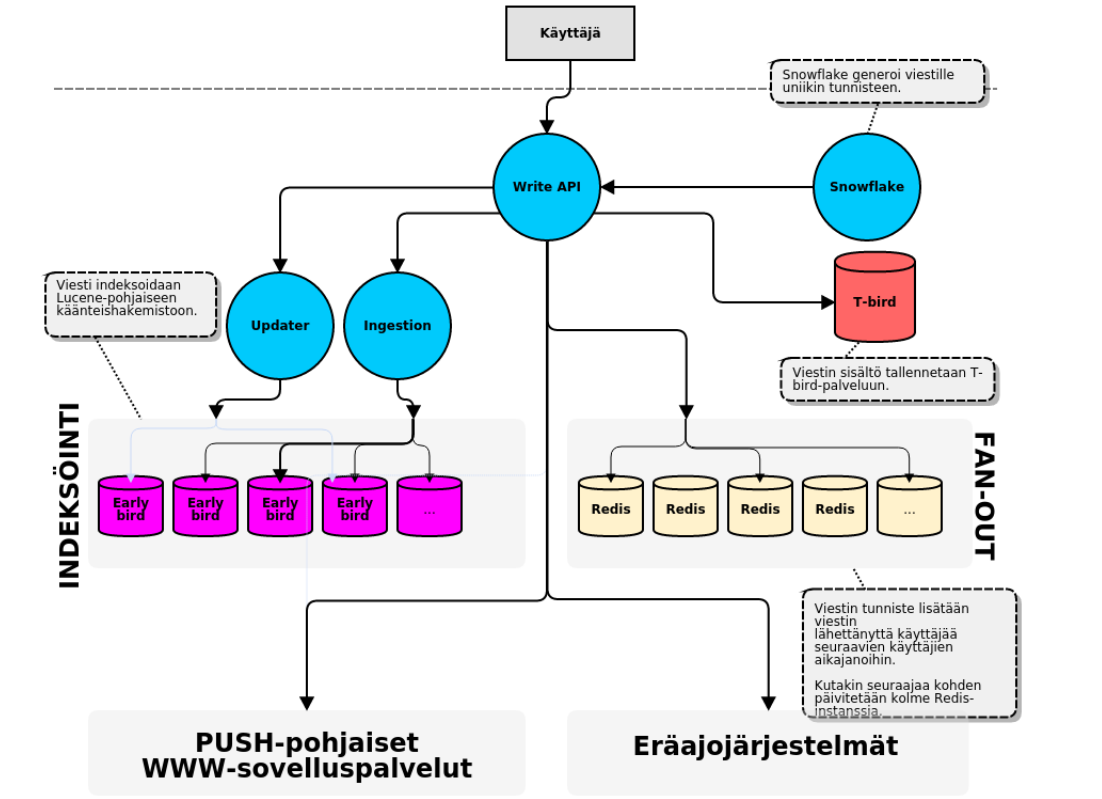
Krikorian [2012a] kuvailee Monorail-arkkitehtuuria siten, että käyttäjältä tuleva pyyntö päätyi ensin kuormantasaajalle, josta se päätyi yksittäiselle Unicorn HTTP-palvelimen [Unicorn, 2015] prosessille, joka käsitteli koko pyynnön kokonaisuudessaan. Krikorian mainitsee, että tallennusjärjestelmä oli eriytetty erilleen, mutta muuten pyynnöt käsiteltiin käytännössä yhden sovelluksen toimesta.

Twitter on 2010-luvun aikana siirtynyt yhden sovelluksen arkkitehtuurista kohti *palvelukeskeistä arkkitehtuuria* (**S**ervice **O**riented **A**rchitecture), mahdollistaakseen sekä paremman suorituskyvyn että myös kehitystyön skaalautuvuuden [Krikorian, 2014].

3.1.3.2 Arkkitehtuuri 2010-luvulla

Twitterin toiminnot rakentuvat uusien viestien vastaanottamisen sekä näiden viestien käyttäjille välittämisen ympärille. Kuvassa 3.3 on kuvattu Twitterin tietovuota tilanteessa, jossa käyttäjä luo uuden viestin. Tietovuokaavio pohjautuu

Krikorianin [2012a, 2012b, 2014] ja Degenhardtin [2014] esitelmiin Twitterin arkkitehtuurista sekä Twitterin kehitystiimin blogikirjoitukseen [Twitter, 2011] Twitterin hakuarkkitehtuurista.



Kuva 3.3 Tietovuokaavio uuden Twitter-viestin käsittelystä.

Kuvassa 3.3 esitetyllä tavalla viesti saapuu käyttäjän asiakasohjelmistolta (selain, mobiiliasiakasohjelma tai muu ohjelmisto) Twitterin *kirjoitusrajapinnalle* (Write API). Kirjoitusrajapinnalta viesti välitetään Twitterin erinäisille sisäisille palveluille, joista on seuraavissa aliluvuissa kuvattu oleellisimmat.

3.1.3.3 Viestien tallentaminen

Viestin tallentamisen alkuvaiheilla Twitter generoi viestille uniikin tunnisteen. Viesti tunnistetaan 20 tavun mittaisella tunnisteella. Tunnisteen ensimmäiset 8 tavua sisältävät viestin yksilöivän uniikin tunnisteen. Seuraavat 8 tavua sisältävät tunnisteen, jolla yksilöidään viestin luonut käyttäjä. Viimeiset neljä tavua sisältävät viestiin liittyvää metadataa, kuten esimerkiksi tiedon siitä, onko kyseessä uudelleentwiittaus. [Krikorian, 2012a]

Alkuaikoina Twitter käytti MySQL:n auto increment -toimintoa viestien tunnisteiden generointiin. Tämä ratkaisu ei kuitenkaan ollut skaalautuva Twitterin mittakaavassa [Krikorian, 2012a]. Twitter siirtyi sittemmin käyttämään itse kehittämäänsä, *Snowflake*-nimistä hajautettua palvelua uniikkien tunnisteiden generointiin. Snowflake generoi 64-bittisiä tunnisteita, jotka koostuvat aikaleimasta, Snowflake-instanssin tunnisteesta sekä järjestysnumerosta. [Twitter, 2015d] Twitterin tapa generoida uniikki tunniste muistuttaa Universally Unique Identifier -tunnisteen (UUID) generointia, vaikkakin Twitterin generoima 64-bittinen tunniste on kooltaan 128-bittistä UUID:tä pienempi [Leach *et al.*, 2005].

Koska Twitterissä tapahtuu merkittävästi enemmän luku- kuin kirjoitusoperaatioita, on arkkitehtuuri optimoitu ennen kaikkea lukuoperaatioiden suorittamista silmällä pitäen. Tämä näkyy esimerkiksi aikajanojen päivittämisessä, joka tehdään aina heti viestiä tallennettaessa, jolloin raskaalta prosessoinnilta vältytään myöhemmin, kun aikajana generoidaan ja esitetään käyttäjälle.

3.1.3.4 Kotisivun aikajanan päivittäminen ja noutaminen

Aktiivisten käyttäjien kotisivujen aikajanat tallennetaan Redis-ryppääseen. Redis on kokonaan keskusmuistissa toimiva avain-arvo-tietokanta. Kukin aikajana tallennetaan kolmelle Redis-ryppään instanssille. [Krikorian, 2012a] Tällä suojaudutaan yksittäisten instanssien rikkoutumisia vastaan. Aktiivisiksi käyttäjiksi tulkitaan ainoastaan ne käyttäjät, jotka ovat käyttäneet Twitteriä viimeisen kuukauden aikana. Redis-rypäs sisälsi vuonna 2013 Krikorianin mukaan joitain teratavuja keskusmuistia. [Krikorian, 2012b]

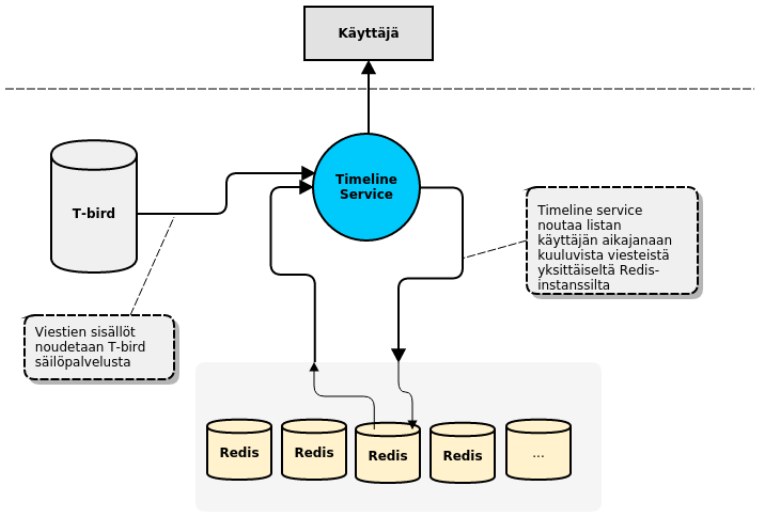
Mikäli käyttäjä ei ole sisäänkirjautuessaan aktiivinen, eli hänen aikajanaansa ei ole tallennettu Redis-instansseille, joutuu Twitter etsimään käyttäjän seuraamat käyttäjät sekä koostamaan näiden julkaisemista viesteistä aikajanan, jonka se palauttaa käyttäjälle sekä tallentaa Redis-instansseille [Krikorian, 2012a].

Redis-instansseille ei tallenneta varsinaisia viestejä, vaan ainoastaan viestien tunnisteet, joiden avulla viestien sisällöt voidaan myöhemmässä vaiheessa koostaa sopivan muotoisiksi eri palveluja hyödyntäen. Twitterissä luotujen viestien sisällöt tallennetaan *T-bird*-nimiseen palveluun.

Aina kun uusi viesti luodaan, lisätään viittaus kyseiseen viestiin kaikkien viestin luonutta käyttäjää seuraavien käyttäjien kotisivuaikajanoihin. Käytännössä tämä tarkoittaa kunkin käyttäjän kohdalla kolmen Redis-instanssin päivittämistä. Twitter käyttää tästä termiä *hajaannuttaminen* (fan-out).

Koska uusi twiitti kirjoitetaan aina kaikkien aktiivisiksi tulkittujen seuraajien

aikajanoihin, on kirjoitusoperaation aikavaativuus $\mathcal{O}(n)$, missä n on seuraajien lukumäärä. Toisaalta koska kotisivujen aikajanat ovat koko ajan Redis-instanssien keskusmuistissa, ei kotisivun aikajanan generointia varten ole tarve tehdä mitään aikavaativuudeltaan monimutkaista. Aikajana saadaan generoitua, kun etsitään yksi niistä kolmesta Redis-instanssista, jolle käyttäjän aikajana on tallennettu. Redis-instanssilta otetaan talteen lista viestien tunnisteista ja näiden pohjalta noudetaan varsinaiset viestit T-bird-palvelusta. Aikajanan generoinnin tietovuota on kuvattu kuvassa 3.4.



Kuva 3.4 Tietovuokaavio kotisivun aikajanan generoinnista käyttäjälle.

3.1.3.5 Hakutoiminto

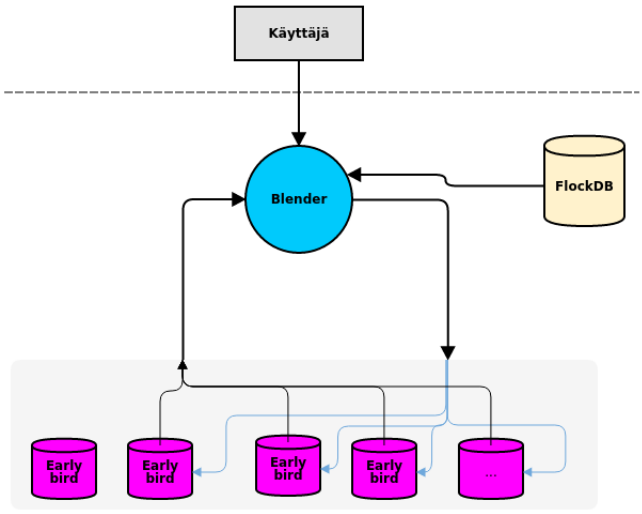
Toinen palvelu, johon viestit viedään heti luomisen jälkeen on Twitterin hakutoiminnon taustalla oleva käänteishakemistoon (reverse index) pohjautuva indeksointipalvelu. Twitter indeksoi viestit myöhemmin tapahtuvia hakuja varten käyttämällä heidän itse kehittämäänsä *Earlybird*-ohjelmistoa. *Earlybird*-ohjelmisto rakentuu Lucene-kokotekstihakukirjaston (full-text search library) [Lucene, 2015] päälle. [Busch *et al.*, 2012]

Twitter mukauttaa hakutuloksia käyttäjäkohtaisesti ottamalla huomioon esimerkiksi hakua suorittavan käyttäjän sosiaalisen verkoston. Hakua suorittavan käyttäjän seuraamien käyttäjien viestit saavat haussa korkeamman pisteytyksen ja ovat siten todennäköisemmin mukana hakutuloksissa. Twitter pyrkii parantamaan hakutuloksia poistamalla niistä moninkertaistetut viestit, joita ilmaantuu

varsinkin silloin, kun ihmiset twiittaavat suosituista tapahtumista. [Twitter, 2011]

Twitter käyttää ennen Earlybirdeille tehtävää tallennusta tapahtuvasta esikäsittelystä nimeä *Ingestion*. Esikäsittelyvaiheessa viestit tokenoidaan ja niihin lisätään metadataa, kuten tieto käyttäjän käyttämästä kielestä. *Updater*-niminen palvelu välittää edelleen hakuindekseihin päivityksiä indeksien luonnin jälkeen, ja mahdollistaa näin hakutulosten mukauttamisen esimerkiksi käyttäjien suorittamien aktiviteettien mukaan [Twitter, 2011].

Hakua suoritettaessa, käyttäjän hakupyyntö päätyy *Blender*-nimiselle palvelulle, joka näkyy kuvan 3.5 tietovuokaaviossa. Blender välittää hakutermit usealle Earlybird-palvelimelle samanaikaisesti [Krikorian, 2012a]. Earlybird-palvelimet käsittelevät pyynnöt toistensa kanssa rinnakkain. Hakutulokset palautuvat Blender-palvelulle, joka tarkentaa hakutuloksia esimerkiksi poistamalla niistä moninkertaistetut esiintymät [Twitter, 2011].



Kuva 3.5 Tietovuokaavio käyttäjän suorittaman haun käsittelystä.

Twitter *sirpaloi* (shard) jokaisen viestin tietylle joukolle Earlybird-instansseja laskemalla viesteille hajautusarvot (hash) ja käyttämällä näitä hajautusarvoja osituksen perustana (hash partitioning) [Twitter, 2011].

3.1.3.6 Muut palvelut

Aikajanojen ja indeksoinnin päivittämisen lisäksi käyttäjän luoma uusi viesti viedään heti myös sekä WWW-sovelluspalvelujärjestelmälle että eräajojärjestelmälle.

Eräajojärjestelmä vastaa vähemmän reaaliaikaisesta prosessoinnista. Twitteristä on mahdollista saada esimerkiksi yhteenvetosähköposteja viime aikoina julkaistuista viesteistä. Eräajojärjestelmä vastaa muun muassa tästä toiminnallisuudesta [Krikorian, 2012a].

WWW-sovelluspalvelujen (Streaming API ja Firehose) vastuulla on viestien välittäminen lähes reaaliaikaisesti asiakasohjelmistoille sekä muille osapuolille, jotka tahtovat vastaanottaa Twitteriin tallennettua dataa.

Edellä mainittujen piirteiden ohella Twitterin taustalta löytyy lisäksi vielä monia muita toimintoja. Twitter muun muassa varmuuskopioi dataa taustalta löytyvään HBase-tietokantaryppääseen. Lisäksi Twitter käyttää HBaseä muun muassa käyttäjähaun taustalla sekä aikasarjamuotoisen datan tallentamiseen. [HBase, 2015e]

3.1.3.7 Erikoistapaukset

Suositut käyttäjät muodostavat Twitterissä poikkeuksen. Tällaisia käyttäjiä ovat esimerkiksi aiemmin mainittu Kate Perry, jolla on lähemmäs 70 miljoonaa seuraajaa. Mikäli esimerkiksi Kate Perryn kohdalla suoritettaisiin normaali hajaannuttamisoperaatio (fan-out), jossa kaikkien seuraajien aikajanat päivitettäisiin heti, tarkoittaisi tämä kaikkineen lähemmäs 70 miljoonan kirjoitusoperaation suorittamista. Ja koska kukin aikajana on replikoitu kolmelle Redis-instanssille, johtaisi tämä kaikkineen lähemmäs 210 miljoonaan kirjoitusoperaatioon.

Krikorian [2012a] kuvaileekin, että Kate Perryn kaltaisten hyvin suosittujen henkilöiden kohdalla voi olla parempi jättää hajaannuttaminen tekemättä, ja sen sijaan lisätä suosittujen käyttäjien viestit heidän seuraajiensa aikajanoihin vasta siinä vaiheessa, kun aikajanoja generoidaan näytettäväksi.

3.1.3.8 Huomioita

Twitterin arkkitehtuuri on suunnattu tarkasti tietyn tehtävän suorittamiseen: niiden viestien vastaanottamiseen, joita käyttäjän seuraamat käyttäjät lähettävät. Suuri osa Twitterin arkkitehtuurista rakentuu sen ympärille, että kotisivujen aikajanat saataisiin päivitettyä tehokkaasti ja reaaliaikaisesti. Tämä johtaa kuitenkin tiettyihin rajoitteisiin sen suhteen, miten Twitteriä on mahdollista käyttää. Joissain tilanteissa voisi olla kätevämpää seurata tiettyjä aihetunnisteita kuin tiettyjä ihmisiä. Tämä ei kuitenkaan onnistu muuten, kuin etsimällä kyseisiä aihetunnisteita hakutoiminnon avulla. Aikajanojen reaaliaikainen päivittäminen olisi todennäköisesti monimutkaisempaa, jos käyttäjien olisi mahdollista seurata myös aihe-

tunnisteita. Erittäin suosittuja käyttäjiä on Twitterissä suhteellisen pieni määrä. Erittäin suosittuja aihetunnisteita saattaisi kuitenkin olla hyvin paljon suurempi määrä.

Twitter asettaa myös joitain muita rajoitteita. Lähetettyjä viestejä ei ole esimerkiksi mahdollista muokata jälkikäteen. Twitter eroaa tämän osalta esimerkiksi perinteisistä keskustelufoorumeista, joissa viestien muokkaaminen jälkikäteen on yleisesti mahdollista. Toisaalta Twitter toimii tältä osin vastaavaan tapaan kuin tekstiviestien lähetys kännykällä tai sähköpostiviestien lähettäminen tietokoneella toimii. Ja tämä on osaltaan myös luonnollista, koska yksi tapa lähettää viestejä Twitteriin on tekstiviestin lähettäminen Twitterin tarjoamaan puhelinnumeroon.

3.2 Reddit

3.2.1 Yleinen kuvaus

Reddit on linkkien, uutisten ja keskusteluiden jakamiseen suuntautunut sosiaalisen median palvelu. Redditissä on mahdollista julkaista tekstimuotoisia *viestejä*. Viestit voivat olla WWW-linkkejä toisille sivustoille (usein kuvalinkkejä) tai tekstimuotoisia viestejä. Viestin julkaiseminen muodostaa uuden *viestiketjun* (thread). Viestiketjussa muut käyttäjät voivat *kommentoida* (comment) alkuperäisen viestin sisältöä. Viestiketjujen sekä viestiketjuissa esiintyvien kommenttien vieressä näkyy aina *plus-* ja *miinusäänestyspainikkeet* (up/down voting). Kukin käyttäjä voi äänestää kutakin viestiketjua ja kommenttia yhden kerran, joko lisäten (up vote) tai vähentäen (down vote) viestiketjulta tai kommentilta yhden pisteen. Pisteistä käytetään yleisesti nimitystä *karmapiste* (karma point).

Viestit ja viestiketjut julkaistaan aina tietyllä ennalta luodulla *aihealueella*. Aihealueista Reddit käyttää nimitystä *subreddit*. Viestiketjua ei luomisen jälkeen voi siirtää aihealueelta toiselle. Aihealue on tietyllä tapaa samantapainen kuin Twitterin aihetunniste. Se liittää viestin tiettyyn aiheeseen. Reddit käyttää aihealueista myös nimitystä *yhteisö* (community), koska aihealueet keräävät usein juuri jostain tietystä aiheesta kiinnostuneita ihmisiä yhteen. Erona Twitterin aihetunnisteisiin on kuitenkin se, että toisin kuin Twitterissä, Redditin aihealueen tulee olla etukäteen perustettu.

Redditissä ei ole mahdollista seurata yksittäisiä käyttäjiä, vaan käyttäjät seuraavat aihealueita. Reddit sallii kuitenkin toisten käyttäjien merkkaamisen ystäviksi, minkä jälkeen näiden käyttäjien avaamia viestiketjuja ja kommentteja on helpompi seurata.

Redditin etusivulla näytetään viestiketjut niiltä aihealueilta, joita käyttäjä seuraa. Oletuksena uudet käyttäjät seuraavat muutamaa kymmentä oletusaihealuetta. Lisäksi jokaisella aihealueella on oma etusivunsa, jolla näytetään vain kyseisen aihealueen sisältämiä viestiketjuja.

Yksittäisen aihealueen seuraamisen sijaan Redditissä on mahdollista luoda myös listoja, joiden avulla on mahdollista luoda uusia "etusivuja", joissa näytetään viestiketjuja kaikista listaan liitetyistä aihealueista. Reddit käyttää näistä käyttäjien luomista listoista nimeä *Multireddit*. [Goodman, 2013]

Mielenkiintoisesti Redditin aihealueen seuraaminen muistuttaa melko läheisesti käyttäjän seuraamista Twitterissä. Vastaavasti Redditin listat muistuttavat monella tapaa Twitterin listoja. Suurimpana erona on se, että toisin kuin Twitterissä, Redditissä seurataan käyttäjien sijaan aihealueita.

Redditin etusivulla sekä kullakin aihealueella on mahdollista lajitella viestiketjut muutamilla eri tavoilla. Yksi vaihtoehto on seurata uusimpia (new-lajittelu) aihealueelle lisättyjä viestiketjuja. Toinen yleinen tapa on seurata suosittuja viestiketjuja (hot-lajittelu). Hot-lajittelu huomioi viestiketjun sijainnissa viestiketjun iän (suosien uudempia ketjuja), annettujen karmapisteiden kokonaismäärän (plus- sekä miinuspisteet) sekä plus- ja miinuspisteiden välisen suhteen.

Käyttäjän julkaisemien viestiketjujen ja kommenttien perusteella kullekin käyttäjälle määritellään käyttäjäkohtaiset karmapisteet. Käyttäjäkohtaiset karmapisteet kertovat käyttäjän aktiivisuudesta sekä siitä, miten muut käyttäjät ovat reagoineet käyttäjän viestiketjuihin sekä kommentteihin. Käyttäjäkohtaiset karmapisteet määrittelevät käyttäjän *mainetta* samantapaisesti kuin Twitterissä seuraajien lukumäärä.

3.2.2 Laadulliset vaatimukset

Vuoden 2015 huhtikuussa Reddit keräsi noin 169 miljoonaa uniikkia kävijää sivustolleen. Aktiivisia aihealueita Reddit listasi toukokuun 2015 alussa hieman vajaat 9500 kappaletta. [Reddit, 2015a] Aktiivisiksi aihealueiksi Reddit laskee ne aihealueet, joille on edellisen päivän aikana lisätty vähintään viisi uutta viestiketjua tai kommenttia [Goodman, 2014].

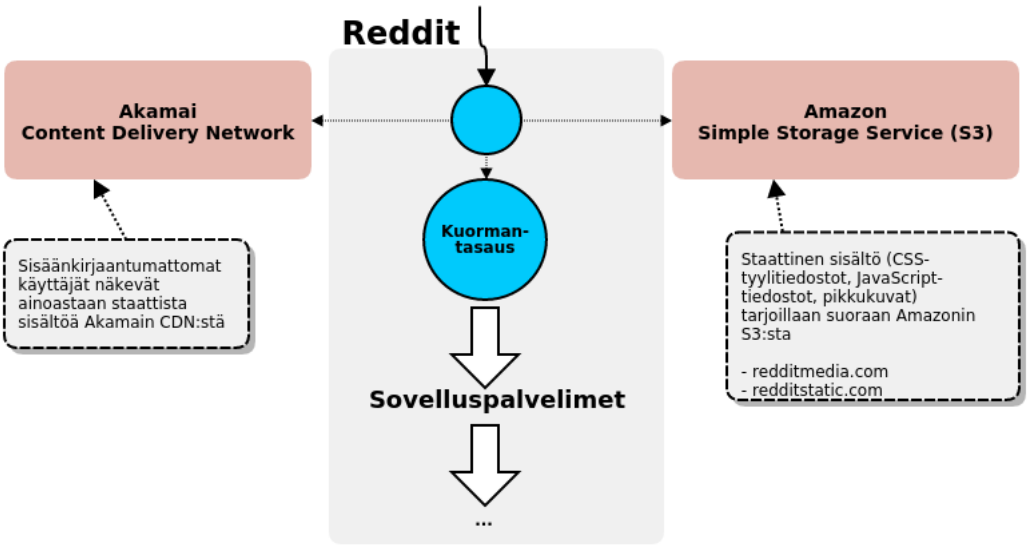
Yksittäisiä sivun katselukertoja tapahtui vuoden 2015 huhtikuussa hieman reilut 7,5 miljardia kappaletta. Plus- ja miinuspisteitä käyttäjät jakoivat tuona aikana kaikkiaan hieman vajaat 25 miljoonaa kappaletta. [Reddit, 2015a]

Redditiin tallennettiin vuonna 2013 hieman yli 40 miljoonaa viestiketjua, 404 miljoonaa kommenttia, sekä kaikkineen 6,7 miljardia plus- ja miinuspistettä. Si-

vujen katselukertoja Reddit keräsi 56 miljardia kappaletta vuonna 2013. [Martin, 2013]

3.2.3 Arkkitehtuuri

Redditissä pyynnön käsittely haarautuu ylimmällä tasolla kolmeen eri haaraan. Haarautuminen tapahtuu sen mukaan, minkälainen pyyntö on kyseessä. Kuvassa 3.6 esitetään tätä tietovuota Huffmanin ja Williamsin [2014] kuvauksen pohjalta. Mikäli pyynnön suorittaa sisäänkirjautumaton käyttäjä, palautetaan vastaus pyyntöön kolmannen osapuolen, Akamain [Akamai, 2015], sisällönjakeluvälimuistista (**C**ontent **D**elivery **N**etwork). Tällaisessa tilanteessa pyynnön käsittely ei vaadi sisällön dynaamista generointia, joten pyyntö ei aiheuta kuormaa Redditin järjestelmälle. Muu staattinen sisältö, kuten pikkukuvat, CSS-tyylitiedostot ja JavaScript-tiedostot, tallennetaan ja palautetaan suoraan Amazonin Simple Storage Servicestä (Amazon S3). Edelleen näistä tiedostoista ei muodostu juurikaan kuormaa Redditin järjestelmälle, vaan kuorman hoitaa kolmannen osapuolen palvelu. Redditin oma järjestelmä huolehtii sisäänkirjautuneille käyttäjille näkyvän, dynaamisen sisällön generoimisesta.



Kuva 3.6 Redditin korkean tason tietovuo Huffmanin ja Williamsin [2013] mukaan.

Reddit käyttää palvelujensa toteuttamiseen Amazon Elastic Compute -pilveä (Amazon EC2). Reddit siirtyi käyttämään EC2:ta vuoden 2009 toukokuussa [Ed-

berg, 2013b]. Harveyn [2014a] mukaan Redditin toiminnasta vastasi vuonna 2014 parhaimmillaan noin 300–400 EC2-instanssia (EC2 instance). Tarkkaa tietoa yksittäisten instanssien suorituskyvystä ei ole, mutta Harvey [2014a] listaa Redditin käytössä olleiden palvelinten lukumäärät seuraavasti:

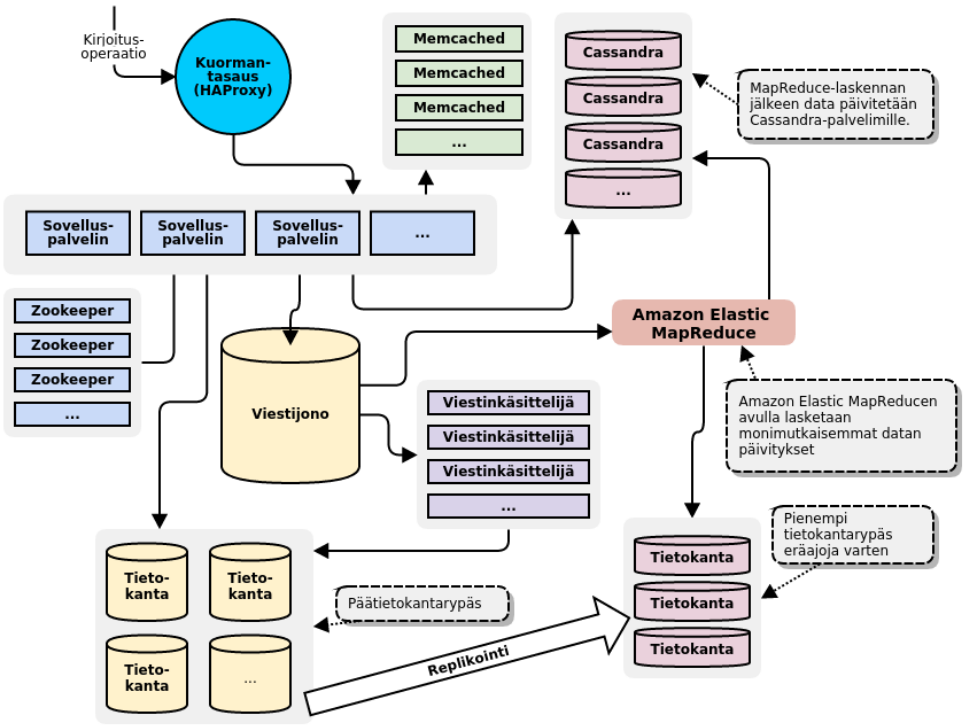
- 230 sovelluspalvelinta
- 73 Memcached-palvelinta
- 16 PostgreSQL-tietokantapalvelinta
- 15 Cassandra-tietokantapalvelinta
- 11 kuormantasauspalvelinta
- 5 eräajopalvelinta
- 30 muuta palvelinta.

Kuvassa 3.7 kuvataan Redditin tietovuota tilanteessa, jossa järjestelmään kirjoitetaan dataa. Kuvaus pohjautuu Huffmanin ja Williamsin [2014] esitykseen.

Kirjoituspyyntö päätyy ensin kuormantasaukseen. Reddit käyttää sovelluspalvelimille suuntautuvaan kuormantasaukseen HAProxyä (**H**igh **A**vailability **P**roxy) [Huffman & Williams, 2014, Edberg, 2013b]. Kuormantasaukselta pyyntö ohjataan sovelluspalvelimille, jotka välittävät päivityksiä suoraan Memcached- sekä Cassandra-palvelimille. Välittömien päivitysten lisäksi kirjoituspyynnöstä voi muodostua prosessointipyyntöjä, jotka tallennetaan viestijonoon.

3.2.3.1 Viestijono

Reddit pyrkii suorittamaan raskaampaa prosessointia vaativat työt viestijonon kautta. Reddit käyttää RabbitMQ-viestijonoa tähän käyttötarkoitukseen [Reddit, 2015b]. Koska käyttäjän lähettämään pyyntöön voidaan lähettää vastaus jo, ennen kuin kaikkea taustaprosessointia on tehty, kyetään käyttäjän pyyntöön vastaamaan nopeammin. Viestijonon käyttäminen toimii myös yhtenä kuormantasauksen muotona. Viestien tallentaminen viestijonoon on kevyempi operaatio kuin itse viestien käsittely. Ruuhkatilanteessa, jossa viestijonossa olevien viestien määrä alkaa kasvamaan, on mahdollista lisätä viestejä käsittelevien instanssien lukumäärää tai odottaa ruuhkan helpottamista.



Kuva 3.7 Redditin tietovuo datan kirjoitustilanteessa Huffmanin ja Williamsin [2014] mukaan.

Reddit käyttää viestijonoa esimerkiksi plus- ja miinuspisteiden päivittämiseen, uusien kommenttien käsittelyyn, pikkukuvien (thumbnail) luontiin sekä hakuindekseille tehtävien päivitysten läpivientiin [Harvey, 2012].

Viestien käsittelyyn Reddit käyttää kahta eri tapaa. Redditillä on käytössään EC2-instansseja, jotka purkavat viestijonoa. Näiden instanssien lisäksi Reddit käyttää Amazonin Elastic MapReduce -palvelua (Amazon EMR) prosessoinniltaan vaativampien töiden suorittamiseen. Tällaisia töitä ovat esimerkiksi Redditin hot-listojen päivitykset. Amazonin Hadoop-pohjainen EMR-palvelu osittaa suoritettavan työn useille eri instansseille ja kokoaa lopuksi lopputuloksen palvelua kutsuneen tahon käytettäväksi. MapReducelta saadut tulokset tallennetaan edelleen Cassandra-instansseille. [Huffman & Williams, 2014]

3.2.3.2 Memcached ja Cassandra

Reddit käyttää sekä avain-arvo-tietokanta Memcachedia että sarakeperhetietokanta Cassandraa välimuisteina, jotta sisältö pystyttäisiin tarjoilemaan käyttäjille nopeammin. Reddit lajittelee esimerkiksi aihealuilla näkyviä viestiketjuja ennalta ja tallentaa näitä listauksia Cassandraan [Huffman & Williams, 2014]. Memcached on kuvattu tarkemmin luvussa 5.10.1 ja Cassandra luvussa 5.10.2.

Cassandra mahdollistaa automaattisesti sekä tallennettavan datan sirpaloimisen että replikoimisen eri pilvi-instansseille. Sirpaloimalla datan eri instansseille, kohdistuu yhdelle instanssille pienempi kuorma ja Cassandra-palvelinryppään suorituskyky on näin mahdollista pitää parempana. Cassandra mahdollistaa myös uusien instanssien lisäämisen ryppääseen sekä jo järjestelmään tallennetun datan jakamisen myös näille uusille instansseille.

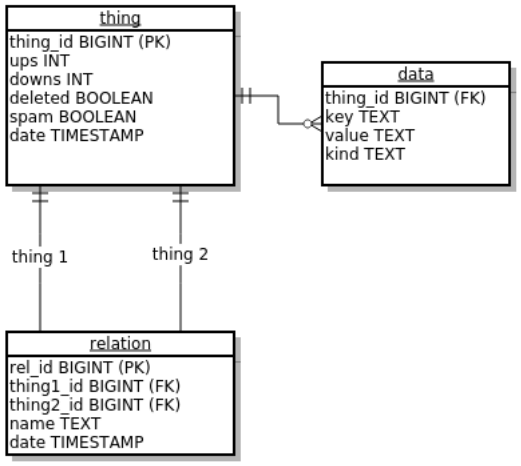
Lisäksi Reddit käyttää hyväkseen Cassandran Bloom-suodattimia (Bloom filter) negatiivisten hakujen (negative lookup) suorittamiseen [Edberg, 2013b]. Bloom-suodattimien avulla voidaan tehokkaasti varmistaa, ettei tietty tietue kuulu tiettyyn etsittyyn joukkoon [Bradberry & Lubow, 2013, 61] Negatiivisia hakuja käytetään esimerkiksi sen selvittämiseen, mitä viestiketjuja tai kommentteja käyttäjä *ei* ole vielä äänestänyt.

3.2.3.3 PostgreSQL ja ThingDB

Reddit käyttää relaatiotietokantanaan PostgreSQL:ää. Redditin käyttämä skeema mahdollistaa avain-arvo-parien tallentamisen tietokantaan. Reddit kutsuu käyttämäänsä skeemaa *ThingDB*-skeemaksi (TDB tai TDB2). Skeemaa koostuu pääosin kolmesta taulusta. Nämä taulut ovat *thing*, *data* ja *relation*.

Lähes kaikki Redditissä käytetyt dataobjektit tallennetaan thing-tauluun. Objekteja ovat esimerkiksi viestiketjut, kommentit, aihealueet ja käyttäjien käyttäjäprofiilit. Thing-taulun rivi sisältää ainoastaan uniikin tunnisteen, plus- ja miinuspisteiden lukumäärät sekä muutaman muun yleisesti käytetyn kentän. Thing-riviin liittyvät avain-arvo-parit tallennetaan data-tauluun, joka koostuu viittauksesta thing-tauluun sekä avain- ja arvokentistä. Koska thing- ja data-taulut eivät määrittele mitä niihin tallennetaan, on skeema avoin uusien objektityyppien sekä niihin liittyvien attribuuttien lisäämiselle.

Thing- ja data-taulujen lisäksi ThingDB-skeemasta löytyy vielä kolmas, *relation*-niminen, tietokantataulu. Relation-taulun avulla on mahdollista linkittää eri thing-rivejä toisiinsa. Relation-taulu toimii vastaavaan tapaan kuin thing- ja data-taulutkin, ja sallii helposti uusien relaatiotyyppien lisäämisen. ThingDB-skeema on kuvattu tietokantakaaviona kuvassa 3.8. Tietokantakaavio pohjautuu Edbergin [2013a] ja Harveyn [2014b] kuvaukseen ja sen tarkoituksena on tarjota mielikuva ThingDB:n skeemasta, ei niinkään kuvata yksityiskohtaisesti kaikkia skeemasta löytyviä tauluja ja sarakkeita.



Kuva 3.8 Tietokantakaavio Redditin ThingDB:stä.

ThingDB:n tyylinen avain-arvo-parien tallentaminen relaatiotietokantaan ei sinällään ole uusi keksintö. ThingDB:n skeemaa kutsutaan yleisesti *entiteetti-attribuutti-arvo-skeemaksi* (**E**ntity-**A**ttribute-**V**alue schema) [Anhøj, 2003].

ThingDB:n etuihin kuuluu hyvä mahdollisuus eri datatyyppien sirpaloimiseen eri tietokantoihin. Reddit käyttää tätä hyväkseen ja vuonna 2013 Redditin tietokanta oli sirpaloitu neljälle isäntä-orja-kokoonpanossa olevalle PostgreSQL-

ryppäälle. Kullakin isännällä oli omat tietokantansa, joihin oli tallennettu eri tyyppisiä dataobjekteja. Kunkin isännän takana oli käytössä yksi tai useampi orjainstanssi, joille pyrittiin ohjaamaan lukuoperaatiot. Redditillä oli käytössään sovelluslogiikkaa, joka pyrki huomaamaan liiallisesti kuormittuneet orjainstanssit ja joka pyrki käyttämään muita vähemmän kuormittuneita orjainstansseja, jos tämä oli mahdollista. [Edberg, 2013b]

Isäntä- ja orjainstanssien väliseen replikointiin Reddit käyttää Londistea [Huffman & Williams, 2014]. Londiste mahdollistaa asynkronisen replikoinnin isäntäinstanssilta yhdelle tai useammalle orjainstanssille [PostgreSQL, 2014].

3.2.3.4 Aktiivinen sisältö

Redditissä sisältö menettää arvoaan ajan myötä. Uudemmat viestiketjut arvostetaan korkeammalle Redditin lajittelualgoritmeissa ja ne päätyvät siten todennäköisemmin eri aihealueiden etusivuille kuin esimerkiksi pari päivän ikäiset viestiketjut.

Koska Redditin aihealueiden etusivuilla näkyvä sisältö on hyvin ajankohtaista ja vain pieni osa käyttäjistä päätyy yleensä vanhemman sisällön pariin, on Redditissä mahdollista tallentaa aktiivinen data pääosin välimuisteihin. Edberg [2013a] kuvailee, että suuri osa, jopa 98 % aktiivisesta sisällöstä, on tallennettu Memcachediin.

Pienentääkseen järjestelmään muodostuvaa kuormitusta Reddit ei salli kommenttien lisäämistä tai plus- ja miinuspisteiden antamista viestiketjuihin, jotka ylittävät tietyn iän [Edberg, 2013b]. Viestiketjut päätyvät tilaan, jossa ainoastaan niiden lukeminen on mahdollista. Lisäksi aihealueita ei voi selata loputtomiin. Aihealueet on jaettu sivuihin, joista jokaisella näkyy 25 viestiketjua. Kunkin aihealueen osalta käyttäjän tavoitettavissa on ainoastaan ensimmäiset 1000 sivua [Huffman & Williams, 2014].

3.2.3.5 Mediatiedostojen käsittely

Redditille on ominaista, että viestiketjussa on linkki kuva- tai videotiedostoon. Reddit ei säilö näitä kuva- tai videotiedostoja itse. Sen sijaan Redditin viestiketjut sisältävät aina vain linkin kolmannen osapuolen järjestelmästä löytyvään kuva- tai videotiedostoon. Tällaisia kolmannen osapuolen palveluja ovat esimerkiksi videoidenjakopalvelu *Youtube* ja kuvienjakopalvelu *Imgur*.

Vaikkei Reddit säilökään varsinaisia mediatiedostoja itse, säilöö se kuitenkin mediatiedostoista luomansa pikkukuvat (thumbnail). Reddit käyttää pikkukuvien

säilömiseen Amazonin Simple Storage Serviceä (Amazon S3). Redditiä selatessa, selain noutaa mediatiedostot suoraan S3:sta. [Edberg, 2013b]

3.2.3.6 Hakutoiminto

Hakutoiminnon Reddit on toteuttanut käyttäen Amazonin CloudSearch-palvelua [Amazon, 2015a]. CloudSearch indeksoi käyttäjiensä osoittaman datan.

CloudSearch skaalaa käyttämiensä pilvi-instanssien määrän ja suorituskyvyn siten, että se mahdollistaa aina sekä suorituskykyisen indeksoinnin että pienen viiveen hakupyyntöihin vastatessa. Datamäärän kasvaessa CloudSearch pyrkii alkuvaiheessa kasvattamaan suorituskykyä siirtämällä datan suorituskykyisemmälle pilvi-instanssille. Jos tämä ei riitä, toteuttaa CloudSearch automaattisesti hakuindeksin sirpaloinnin useammalle pilvi-instanssille. Hakupyyntöjen määrän kasvaessa, CloudSearch replikoi hakuindeksin useammalle rinnakkaiselle pilvi-instanssille, vähentäen näin yhteen instanssiin kohdistuvien hakupyyntöjen määrää. [Amazon, 2015b]

3.2.3.7 Muuta

Tietyissä tilanteissa käyttäjän lähettämän pyynnön käsittely ei onnistu. Redditissä on ollut vuosien varrella yleistä, että käyttäjät ovat aina välillä törmänneet sivuun, joka toteaa, että sivua ei kyetä palauttamaan, koska sen generoinnissa kului liian kauan aikaa. Tähän tilanteeseen voidaan päätyä mikäli sovelluspalvelimilla tapahtuu virhe kyseisen sivun generoinnissa tai mikäli sovelluspalvelimia edeltävillä palvelimilla huomataan, että pyynnön käsittelyssä kestää liian kauan [Williams, 2014].

Redditillä on käytössään kokonaan oma joukko palvelimia Googlen palvelemiseksi. Googlen indeksointibotit pyrkivät indeksoimaan kaiken saatavilla olevan datan, ja suuri osa tästä datasta on sellaista, ettei se ole enää välimuisteissa. Huffman kuvailee Googlen käyttävän sivustoa täysin eri tavoin kuin normaalit käyttäjät, joten sitä varten on järkevää varautua eri tavalla. [Huffman & Williams, 2014]

3.3 Wikipedia

3.3.1 Yleinen kuvaus

Wikipedia on yhteisöllinen tietosanakirja, jonka sisältö on Wikipedian käyttäjäyhteisön luomaa. Wikipedia kuvaa tavoitteekseen olla suurin, kattavin ja par-

haiten saatavilla oleva tietosanakirja [Wikipedia, 2014g]. Wikipedia sai alkunsa vuonna 2001. Wikipedian alkuperäisenä tavoitteena oli nopeuttaa sisällön muodostamista vertaisarvioituun Nupediaan [Wikipedia, 2015a].

Wikipedia rakentuu *sivuista* (page). Suurin osa Wikipedian sisällöstä koostuu *artikkelisivuista* (article). Kunkin artikkelin tehtävänä on tarjota tietosanakirjamuotoinen yhteenveto yhdestä tietystä aiheesta. Kukin artikkeli on kirjoitettu tietyllä kielellä (esimerkiksi englanniksi tai suomeksi). Artikkelit luokitellaan päätasolla kielen mukaan. Esimerkiksi englanniksi kirjoitettu artikkeli on aina osa englanninkielistä Wikipediaa ja suomeksi kirjoitettu artikkeli on aina osa suomenkielistä Wikipediaa. Sama artikkeli ei välttämättä esiinny kaikissa Wikipedian kieliversioissa. Saman artikkelin eri kieliversiot sisältävät linkit muihin kieliversioihin. Artikkelisivujen lisäksi Wikipediasta löytyy muun muassa käyttäjäsivuja, ohjesivuja sekä keskustelusivuja.

Sivut kirjoitetaan *wikiteksti*-nimisellä (wikitext) merkkauskielellä. Wikiteksti mahdollistaa monia eri tapoja, joilla tekstiä voidaan muotoilla, muun muassa yleiset tekstin lihavointi- ja kursivointitoiminnot. Wikiteksti mahdollistaa myös kuvien, matemaattisten kaavojen ja esimerkiksi taulukoiden lisäämisen tekstin sekaan. Wikitekstin sallimat merkkauskäytännöt ovat muodostuneet vuosien varrella pikkuhiljaa sen muotoisiksi kuin mitä ne nykyään ovat. Tämän päivän Wikiteksti-kieltä kuvaillaankin varsin monimutkaiseksi [Brown & Wilson, 2012, 190]. Kielen monimutkaisuuden johdosta Wikitekstin parsiminen ja kääntäminen selaimen ymmärtämälle HTML-kielelle on yksi Wikipedian haastavimpia toimintoja [Brown & Wilson, 2012, 190].

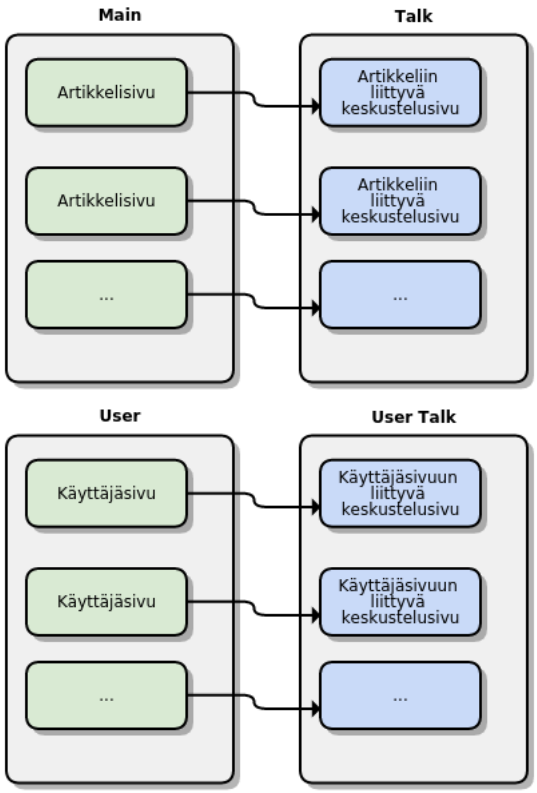
Wikitekstin avulla sivulle on mahdollista myös liittää ennalta määriteltyjä mallinteita (template). Mallinteet mahdollistavat useilla eri sivuilla esiintyvien sisältöjen määrittämiseen yhteen paikkaan ja myöhemmin tämän sisällön sisällyttämisen eri sivuille. Mallinne muodostuu muiden sivujen tapaan wikitekstistä.

Mallinteissa on ollut myös mahdollista käyttää wikitekstin sallimaa ohjelmointikieltä. Tämä melko rajoitettu ja vaikeaselkoinen ohjelmointikieli on sisältänyt yleiset ohjelmointikielissä esiintyvät rakenteet, kuten if- ja else-lausekkeet. Mallinteille on ollut myös mahdollista välittää parametreja, ja siten ohjata niiden sisältämän ohjelmakoodin suorittamista. Vuodesta 2013 lähtien Wikipedia on sallinut Lua-ohjelmointikielen käyttämisen mallinteissaan [Harihareswara, 2013].

Wikipedian sivut sisältävät versiohistorian. Historiasta on mahdollista nähdä jokaisen sivun osalta tiedot kaikista sivuun kohdistuneista muokkauksista. Historia kertoo, kuka muokkauksen on tehnyt, milloin se on tehty sekä sen, miten sivun osia on muokkauksessa muutettu.

Wikipediaan on mahdollista sekä luoda uusia artikkeleita että muokata jo olemassa olevia artikkeleita. Artikkeleiden lisääminen ja muokkaaminen onnistuu joko rekisteröityneenä tai anonyyminä käyttäjänä. Anonyyminä käyttäjänä toimittaessa Wikipedia tallentaa muokkauksen tehneen käyttäjän IP-osoitteen.

Wikipedian sivut on jaoteltu *nimiavaruuksiin* (namespaces). Nimiavaruuksien tehtävänä on erottaa eri aihealueiset sivut toisistaan [Brown & Wilson, 2012, 189]. Artikkelit kuuluvat päänimiavaruuteen (Main-nimiavaruus). Päänimiavaruuden lisäksi Wikipedia sisältää nimiavaruudet esimerkiksi käyttäjä- ja ohjesivuja (User- ja Help-nimiavaruudet) varten. Lähes kaikille nimiavaruuksille löytyy rinnakkainen keskustelunimiavaruus (Talk-nimiavaruus). Rinnakkainen keskustelunimiavaruus tarjoaa paikan, jonka alla Wikipedian käyttäjät voivat keskustella kustakin Wikipedian sisältämästä sivusta. Kaikkineen eri nimiavaruuksia Wikipediasta löytyy 30 kappaletta [Wikipedia, 2014f]. Kuvassa 3.9 näkyy pää- sekä käyttäjänimiavaruudet sekä näitä vastaavat keskustelunimiavaruudet.



Kuva 3.9 Kuvaus Wikipedian nimiavaruuksista ja sivuista.

Nimiavaruuksien sisällä sivuja voidaan edelleen jakaa aihealueisiin liittämällä

sivut *kategorioihin* (category). Kategoriat löytyvät Category-nimiavaruuden alta. Sivu on mahdollista liittää tiettyyn kategoriaan lisäämälle sivulle wikiteksti *[[Category:XYZ]]*. Tämän jälkeen kyseinen sivu listataan automaattisesti kategorian XYZ listaussivulla. Kategorioilla voi olla myös alikategorioita. Tämä mahdollistaa sivujen hierarkkisen luokittelun. [Wikipedia, 2014a]

Tekstimuotoisen sisällön lisäksi mediatiedostot, kuten kuvat, videot, äänitallenteet sekä PDF-tiedostot, muodostavat merkittävän lisän moniin artikkeleihin. Jokaista Wikipediaan tallennettavaa tiedostoa kohden Wikipediaan muodostuu myös uusi tiedostoa vastaava sivu tiedostonimiavaruuteen (File-nimiavaruus). Tiedostoa vastaava sivu sisältää tiedot esimerkiksi tiedoston lisääjästä ja tiedoston käyttöoikeuksista. Artikkeleita varten kuvista luodaan pikkukuvat (thumbnail) tarpeen mukaan. Wikipediaan on mahdollista myös upottaa mediasisältöä Wikipedian ulkopuolelta. Esimerkiksi Wikipedian sisarprojekti, Wikimedia Commons, sisältää yli 24 miljoonaa tiedostoa, joita on mahdollista hyödyntää myös Wikipediassa [Wikimedia, 2015b].

Helpottaakseen sekä ylläpidon kuormaa että parantaakseen artikkelien yhtenäisyyttä, Wikipedia käyttää botteja toistuvien tehtävien suorittamiseen. Botti saattaa esimerkiksi saada herätteen, kun Wikipediaan luodaan uusi artikkeli, ja botti voi tämän herätteen pohjalta tarkistaa, täyttääkö artikkeli Wikipedian asettamat yleiset laatuvaatimukset. Wikipediassa oli vuoden 2015 alkupuolella kaikkiaan 160 aktiivista bottia eri tehtävien suorittamiseen [Wikipedia, 2014d]. Yksittäisellä botilla voi olla useita eri tehtäviä, joita sen kuuluu suorittaa [Wikipedia, 2014c]. Botit muodostavat merkittävän osan Wikipediassa tapahtuvista artikkelien muokkauksista. Englanninkielisen Wikipedian osalta botit ovat suorittaneet kaikkineen kymmeniä miljoonia muokkauksia ja niiden osuus kaikista muokkauksista englanninkielisessä Wikipediassa on noin 12 prosenttia [Wikipedia, 2014e].

Wikipedia tarjoaa lisäksi WWW-sovelluspalvelun (web service, API), jonka avulla Wikipedian toimintoja voidaan hyödyntää ohjelmallisesti esimerkiksi edellä kuvattujen bottien toimesta [Wikipedia, 2015c].

Edellä kuvattujen toimintojen lisäksi Wikipediasta löytyy useita verkkosivuilla yleisesti esiintyviä toimintoja, kuten esimerkiksi hakutoiminto, jonka avulla voidaan etsiä tiettyjä avainsanoja sisältäviä artikkeleita, sekä ATOM- ja RSS-syötteet, joiden avulla on mahdollista seurata esimerkiksi uusien sivujen luontia sekä artikkeleihin tehtyjä muutoksia. Näiden lisäksi Wikipediasta löytyy vielä monia muita toimintoja, joihin useimmat käyttäjät todennäköisesti törmäävät harvemmin.

3.3.2 Laadulliset vaatimukset

Wikipedia eroaa olennaisesti monista muista sosiaalisen median palveluista sillä, että sen toiminnan takana on voittoa tavoittelematon Wikimedia Foundation. Wikipedia kerää rahoituksensa kaikkiin kuluihinsa lahjoituksina. Wikipedia ei näytä sivustollaan mainoksia, eikä se kerää rahaa sijoittajilta. Tämän kaiken takana on Wikipedian tavoite turvata oma itsenäisyytensä sekä riippumattomuutensa. Tämä tarkoittaa sitä, että käytännössä Wikipedialla on selvästi rajoitetumpi määrä varoja käytettävissä infrastruktuuriinsa kuin monilla muilla vastaavan kokoluokan sivustoilla.

Vuonna 2014 Wikipedia-palvelun taustalla oli kaikkineen vain noin 1000 palvelinta [Liambotis, 2014] ja reilut 200 työntekijää [Wikipedia, 2015d]. Vertailun vuoksi vastaavantapaisen määrän kuukausittaisia sivunkatselukertoja keräävä Facebook käyttää palvelunsa ylläpitämiseen ja kehittämiseen arvioiden mukaan toistasataatuhatta palvelinta [Higginbotham & Kern, 2013, Gruener, 2012]. Vastaavasti vuonna 2015 Facebookin palkkalistalla oli yli 8300 työntekijää [Facebook, 2015a].

Lisäksi Wikipedia eroaa esimerkiksi Redditistä ja monista muista toimijoista sillä, että Wikipedia toimii Wikimedia Foundationin itse omistamissa datakeskuksissa. Wikipedia ei käytä ulkopuolisia pilvipalvelutarjoajia, toisin kuin esimerkiksi Reddit. Perusteena tähän on jälleen tarve toiminnan itsenäisyydelle sekä riippumattomuudelle.

Lisäksi Wikipedia pyrkii olemaan hyvin avoin toimintansa suhteen. Wikipedian käyttämät ohjelmistokomponentit ovat avointa lähdekoodia. Samoin Wikipedian infrastruktuuri on kuvattu suhteellisen avoimesti [Liambotis, 2014].

Vuonna 2013 koko Wikipedia keräsi noin 7300 katselukertaa per sekunti. Englanninkielinen Wikipedia kattoi näistä katselukerroista lähes puolet, noin 3500 katselukertaa per sekunti. [Wikimedia, 2015a] Vuonna 2014 Wikipedian arvioitiin keränneen kaikkineen noin 400 miljoonaa uniikkia kävijää joka kuukausi. HTTP-pyyntöjä Wikipedian arvioitiin käsitelleen noin 100000 kappaletta joka sekunti. [Mediawiki, 2015b] HTTP-pyyntöjen määrä vaikuttaa noin kymmenkertaiselta varsinaisiin katselukertoihin verrattuna. Tämä selittynee sillä, että yhden sivun esittämistä varten on yleensä tarve tehdä useampi HTTP-palvelupyyntö tyylitiedostojen, JavaScript-lähdetiedostojen sekä kuvien noutamisen takia. Korkeimmillaan Wikipedia vastasi noin 190000 HTTP-pyyntöön sekunnissa [Liambotis, 2014].

Wikipedia koostuu eri kieliversioista. Esimerkiksi englanninkielinen Wikipedia

löytyy WWW-osoitteesta *http://en.wikipedia.org/*, kun taas esimerkiksi ruotsinkielinen Wikipedia sijaitsee osoitteessa *http://sv.wikipedia.org/*. Kaikkineen eri kieliversiota oli vuoden 2015 alkupuolella 288 kappaletta. Kahdessatoista näistä kieliversioissa oli yli miljoona artikkelia. Viisikymmentä seuraavaksi suurinta kieliversiota sisälsivät nekin kukin yli satatuhatta artikkelia. Toisaalta viisikymmentä pienintä kieliversiota sisälsivät vuoden 2015 alkupuolella kukin alle tuhat artikkelia. [Wikipedia, 2014b]

Kaikkineen eri kieliversioihin on tallennettu yli 34 miljoonaa artikkelia. Mikäli mukaan lasketaan myös muut kuin pelkät artikkelisivut (esimerkiksi keskustelusivut), sisältävät eri kieliversiot kaikkineen yli 128 miljoonaa sivua. Muokkauksia Wikipediaan on tallennettu kaikkineen lähes 2 miljardia kappaletta. [Wikipedia, 2014b]

Suosituin kieliversio, englanninkielinen Wikipedia, sisälsi syyskuussa 2014 noin 4,7 miljoonaa artikkelia. Näitä artikkeleita oli ajan saatossa muokattu yhteensä yli 744 miljoonaa kertaa. Katselukertoja englanninkielisen Wikipedian sivuille kertyi 8,7 miljoonaa kappaletta tunnissa. [Wikipedia, 2014h, Wikimedia, 2015d] Vertailun vuoksi, suomenkielisestä Wikipediasta artikkeleita löytyi syyskuussa 2014 hieman yli 356 tuhatta kappaletta ja katselukertoja per tunti oli hieman vajaat 72 tuhatta kappaletta [Wikimedia, 2015d]. Erot sekä artikkelimäärissä että sivujen katselukerroissa voivat siis olla yli satakertaiset eri kieliversioiden välillä.

Englanninkieliseen Wikipediaan luotiin vuosina 2013–2014 noin 800 uutta artikkelia joka päivä. Muokkauksia englanninkielinen Wikipedia keräsi vuosina 2013–2014 joka kuukausi 2,8–4,8 miljoonaa kappaletta. [Wikimedia, 2015e]

Tekstimuotoisten sivujen ohella Wikipediaan tallennetaan mediatiedostoja, kuten kuva-, ääni- ja videotiedostoja. Vuonna 2014 Wikipediaan oli tallennettu 30 miljoonaa tiedostoa sekä 320 miljoonaa näistä generoitua pikkukuvaa (thumbnail) [Liambotis, 2014].

Yllä esitetyistä numeroista nähdään, että Wikipediassa, kuten Twitterissä ja Redditissäkin, tapahtuu merkittävästi enemmän luku- kuin kirjoitusoperaatioita. Vastaavaan tapaan kuin Twitterin ja Redditin kohdalla, myös Wikipediassa suorituskyvyn optimointi on painottunut erityisesti lukuoperaatioiden suorituskyvyn optimointiin.

3.3.3 Arkkitehtuuri

Wikipedian ensimmäinen versio käytti Perl-pohjaista *UseModWikiä* alustanaan. Wikipediaa ajettiin alkuvaiheessa yhdellä palvelimella ja UseModWikin tieto-

kanta oli tiedostopohjainen. UseModWikin rajoitteet tulivat kuitenkin nopeasti vastaan sivuston suosion kasvettua ja Wikipedian kehitystiimi siirtyi käyttämään heidän itse kehittämäänsä PHP-skriptiä vuoden 2002 alkupuolella. Tätä PHP-skriptiä jatkokehitettiin vuosien varrella ja siitä syntyi lopulta avoimen lähdekoodin Mediawiki-niminen ohjelmisto, joka on tarkasti räätälöity vastaamaan Wikipedian tarpeisiin. [Brown & Wilson, 2012, 180] Tämän päivän Wikipedia rakentuu MediaWiki-ohjelmiston ympärille.

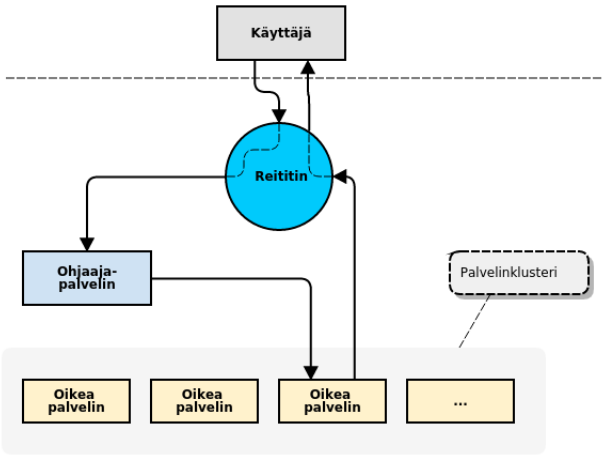
Arkkitehtuurin kuvaus pohjautuu Wikipedian infrastruktuurista vastanneiden Ryan Lanen [2011], Aaron Schulzin [2014] sekä Faidon Liambotisin [2014] esitelmiin Wikipedian toiminnasta sekä arkkitehtuurista. Lisäksi kuvaus pohjautuu Wikipedian itsensä julkisesti ylläpitämään dokumentaatioon, joka löytyy WWW-osoitteesta: `https://wikitech.wikimedia.org/wiki/Main_Page`.

3.3.3.1 Kuormantasaus

Ensimmäinen askel Wikipedialle lähtevän pyynnön käsittelyssä on *nimikyselypyyntö* (domain name server query) *nimipalvelimelle* (domain name server), joka kertoo, mihin IP-osoitteeseen HTTP-palvelupyyntö pitäisi lähettää. Wikipedia suorittaa ensimmäisen askeleen kuormantasauksessa tällä tasolla [Liambotis, 2014]. Wikipedialla on käytössään kolme auktoritatiivista nimipalvelinta (authoritative name server), joiden avulla nimikyselypyynnöt käsitellään [Wikitech, 2014a]. Nimipalvelimilla on käytössä *gdnsd*-niminen ohjelmisto, joka yhdessä geoip-lisäosan kanssa muodostaa nimikyselypyynnöille vastaukset sen mukaan, miltä maantieteelliseltä alueelta pyynnöt ovat peräisin [Liambotis, 2014, Wikitech, 2014a]. Varsinaiset HTTP-palvelupyynnöt pyritään ohjaamaan maantieteellisesti lähimmälle datakeskukselle, jotta pyyntö kyettäisiin käsittelemään mahdollisimman nopeasti.

Vuoden 2014 lopulla Wikipedialla oli käytössä kaikkiaan viisi eri sijaintia palvelinryppäilleen. Wikipedia käyttää palvelinryppäistään nimiä *eqiad*, *codfw*, *ulsfo*, *esams* ja *knams*. Näistä kolmen ensimmäistä on fyysiseltä sijainniltaan USA:ssa ja loput kaksi Amsterdamissa, Hollannissa. [Liambotis, 2014] *Eqiad*-ryppään vastuulla on Wikipedian primääritoiminnot, kuten MediaWiki-ohjelmiston suorittaminen, hakupalvelujen tuottaminen ja muut Wikipedian tarjoamat palvelut. Codfw-rypäs tarjoaa samoja palveluja kuin eqiad-rypäskin, ja toimii sen ohella eqiadin korvaajana mahdollisissa vikatilanteissa. Ulsfo- sekä esams-ryppäät toimivat välimuisteina. Knamsin ainoana tehtävänä on verkkoyhteyksien tarjoaminen esams-ryppäälle. [Wikimedia, 2014]

Datakeskuksessa pyyntö päätyy *LVS-DR*-kuormantasaajalle (**L**inux **V**irtual **S**erver **D**irect **R**outing) [Wikitech, 2014d]. LVS-DR-kuormantasauksessa pyyntö saapuu ensin reitittimelle, josta se ohjautuu *ohjaajapalvelimelle* (Director). Ohjaajapalvelin valitsee taustalla olevasta palvelinryppäästä yhden *oikean palvelimen* (real server), jolle se ohjaa pyynnön käsiteltäväksi. Oikea palvelin käsittelee pyynnön ja palauttaa pyynnön suoraan asiakkaalle, sivuuttaen ohjaajapalvelimen. [Kopper, 2005] Pyynnön käsittelyä on kuvattu kuvassa 3.10.

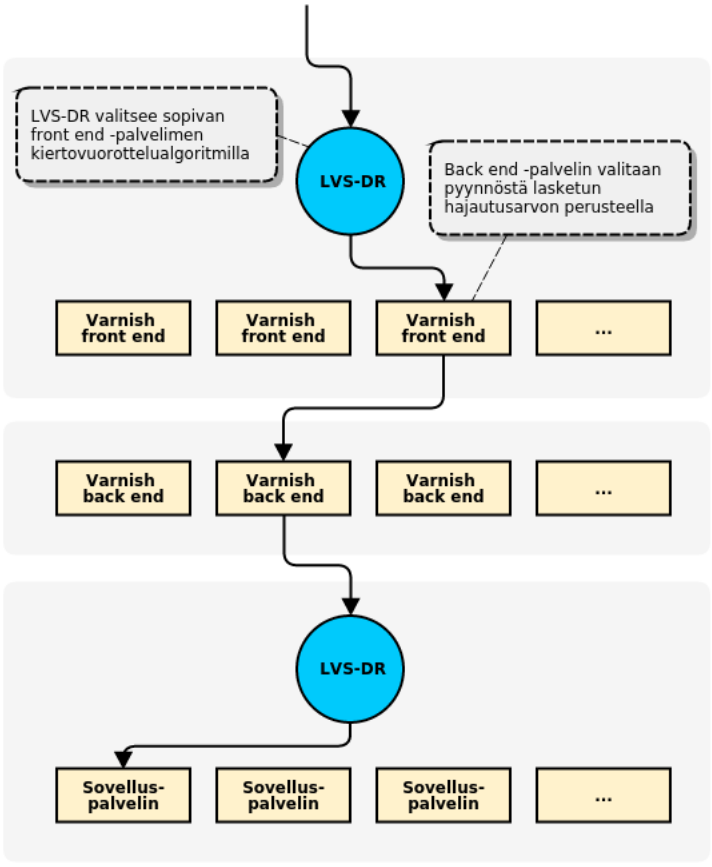


Kuva 3.10 Pyynnön käsittely LVS-DR-kuormantasauksella.

3.3.3.2 Pyynnön käsittely ja datan tallennus

Wikipedia käyttää Varnish-ohjelmistoa *edustapalvelimillaan* (reverse proxy). Varnish-palvelimet toimivat HTTP-välimuisteina (caching HTTP reverse proxy). LVS-DR välittää pyynnön ensin Varnish frontend -palvelimelle yksinkertaisella *kierto-vuorottelualgoritmilla* (round robin algorithm). Varnish frontend -palvelin muodostaa pyynnöstä *johdonmukaisen hajautusarvon* (consistent hash, kuvattu luvussa 5.4.1). Johdonmukaisen hajautusarvon perusteella valitaan Varnish backend -palvelin, jolta pyynnössä etsitty data parhaassa tapauksessa löytyy. Mikäli Varnish backend -palvelin ei sisällä haettua dataa, ohjataan pyyntö edelleen sovelluspalvelimille. [Wikitech, 2014c, Liambotis, 2014] Edustapalvelinten kuormantasausta on kuvattu kuvassa 3.11.

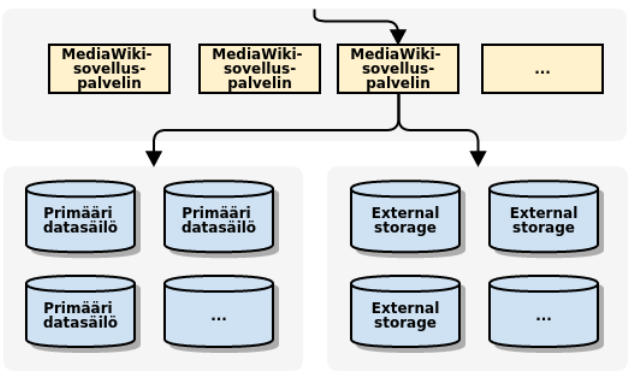
Mikäli haettua sivua ei löydy edustapalvelimen välimuistista, on se tarve generoida. Sivujen generointi tapahtuu MediaWiki-ohjelmiston avulla.



Kuva 3.11 Varnish-edustapalvelinten kuormantasaus.

MediaWiki-ohjelmisto vaatii taustalleen relaatiotietokannan, johon se tallentaa wikiin tallennetun datan. Wikipedia käyttää *primäärinä datasäilönään* (primary datastore) MySQL-taustaista MariaDB:tä. Jokainen Wikipedian kieliversio sijaitsee omassa tietokannassaan [Lane, 2011]. Wikipedian primääri datasäilö on sirpaloitu seitsemään eri tietokantapalvelinryppääseen. Yhden Wikipediakieliversion koko tietokanta sijaitsee aina yhdessä tietokantapalvelinryppäässä. [Liambotis, 2014] Tämänkaltaisella sirpaloimisella voidaan esimerkiksi estää englanninkieliseen Wikipediaan kohdistuvaa kuormitusta aiheuttamasta vahinkoa muiden kieliversioiden Wikipedioille. Kukin rypäs muodostuu 6–11 palvelimen isäntä-orja-kokoonpanosta. Konfiguraatiossa yksi palvelin toimii isäntänä ja loput orjina. [Liambotis, 2014] Isäntä-orja-kokoonpanon toimintatapaa on kuvattu tarkemmin luvussa 5.7.

Primäärin datasäilön ohella Wikipedialla on toinen joukko relaatiotietokantapalvelimia, joista se käyttää nimitystä external storage (ES). Wikipedia käyttää external storagea wikitekstien tallentamiseen [Wikitech, 2014b] [Brown & Wilson, 2012, 184]. External storage on sirpaloitu datan iän mukaan. External storage koostuu useasta isäntä-orja-kokoonpanossa olevasta tietokantaryppäästä. Kukin rypäs koostuu yhdestä isäntäpalvelimesta sekä pääosin kahdesta orjapalvelimesta [Schultz, 2014]. Kaikki muut paitsi viimeisimpänä käyttöön lisätty rypäs ovat tilassa, jossa niiltä voidaan ainoastaan lukea dataa. Kirjoitukset suoritetaan viimeisimpänä lisätyn ryppään isäntäpalvelimelle. Lisäämällä tietyin väliajoin uuden ryppään käyttöön, Wikipedia kykenee sirpaloimaan tietokannan datan iän mukaan. [Wikitech, 2014b] MediaWikin taustalta löytyvät datasäilöt on kuvattu kuvassa 3.12.



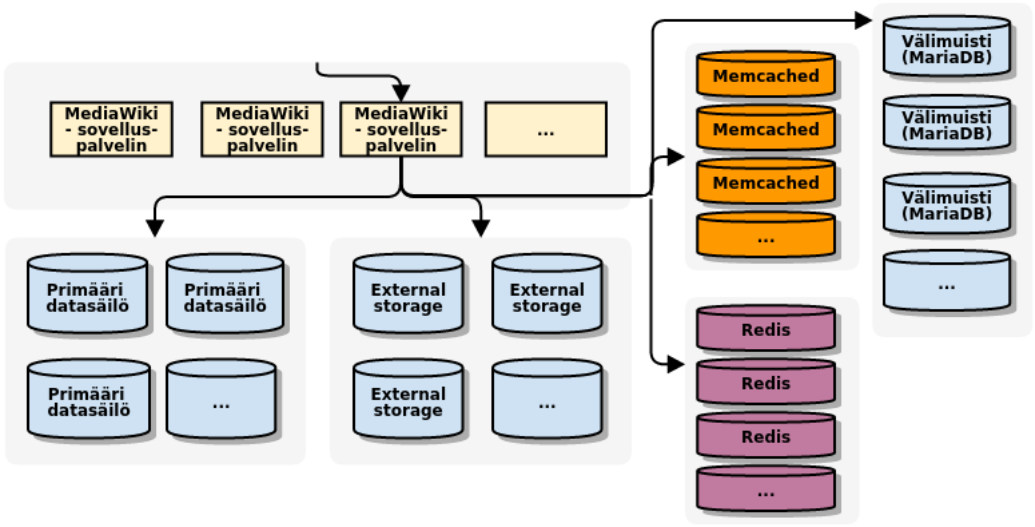
Kuva 3.12 Wikipedian käyttämät relaatiotietokantapohjaiset datasäilöt.

Kuormantasauksen ja isäntä-orja-kokoonpanojen ohella tämän päivän Wikipedia käyttää erityisesti eri välimuistiratkaisuja kyetäkseen palvelemaan kaikkia käyttäjiään rajallisilla resursseillaan. MediaWiki-ohjelmisto on kykenevä hyödyntämään erilaisia välimuistiratkaisuja. [Mediawiki, 2015a] Wikipedia käyttää tätä hyväkseen ja käyttää MediaWiki-ohjelmiston taustalla välimuistina Memcachedia [Liambotis, 2014]. MediaWiki-ohjelmisto tallentaa Memcachediin objekteja, joita se on muodostanut tietokantahakujen pohjalta. Välimuistin tavoitteena on tallentaa usein tarvittuja objekteja, ja siten pienentää sekä taustalla oleviin tietokantoihin että itse MediaWiki-palvelimiin kohdistuvaa kuormaa.

Memcached toimii täysin keskusmuistin varassa. Ollakseen kykenevä toipumaan mahdollisista virhetilanteista paremmin, Memcachedin sisältämiä objekteja varastoidaan myös erilliseen tietokantapalvelinryppääseen. [Schultz, 2014]

Memcachedin ohella Wikipedia käyttää Redistä. Wikipedia tallentaa Redis-instansseille muun muassa MediaWiki-ohjelmiston työjonot (job queue). Lisäksi Wikipedia tallentaa Redikseen esimerkiksi käyttäjien istuntotiedot sekä järjestelmän generoimat lokiviestit [Schultz, 2014, Wikitech, 2014g].

Wikipedia käyttää MediaWiki-ohjelmiston taustalla siis kaikkineen viittä eri datasäilöä. Nämä säilöt on kuvattu kuvassa 3.13.



Kuva 3.13 Wikipedian käyttämät relaatiotietokantapohjaiset sekä muistinvaraiset datasäilöt.

MediaWiki-ohjelmisto itsessään on melko monoliittinen. Varsinkin aiemmin se on pyrkinyt suorittamaan itsenäisesti kaikki Wikipedian kannalta tarpeelliset teh-

tävät. Tämä tilanne on kuitenkin jonkin verran muuttunut, ja Wikipedia käyttää tätä nykyä esimerkiksi wikitekstin HTML:ksi muuntamiseen erillistä, Parsoid-nimistä, www-sovelluspalvelua [Wikitech, 2015]. Wikitekstin parsiminen on monimutkainen ja raskas operaatio. Pitkien ja monimutkaisten artikkelien osalta yksi muuntokerta wikitekstistä HTML:ksi voi viedä useita kymmeniä sekunteja [Wicke, 2013], joten tällaisen raskaan ja hitaan palvelun siirtäminen muista toiminnoista erilleen on ollut luonnollista.

3.3.3.3 Mediatiedostojen käsittely

Relaatiotietokantojen sekä muistinvaraisten avain-arvo-säilöjen lisäksi Wikipedia käyttää erillistä OpenStack Swift -järjestelmää (myöhemmin vain Swift) Wikipediaan tallennettujen mediatiedostojen säilömiseen. [Wikitech, 2014f] Swift kuvaa itseään hajautetuksi, lopulta yhtäpitäväksi (eventually consistent) objektisäilöksi. Myös Swift on MediaWiki-ohjelmiston tavoin sekä LVS-DR-kuormantasauksen että Varnish-edustapalvelinten takana [Wikitech, 2014f].

Järjestelmän luotettavuuden kasvattamiseksi jokainen mediatiedosto replikoidaan kolmelle Swift-palvelimelle. Tarvittaessa Swift-järjestelmästä on mahdollista pyytää viimeisin versio mediatiedostosta. Tällöin Swift käy läpi ne kolme palvelinta, joille kyseinen tiedosto on tallennettu ja palauttaa näistä kaikista ajantasaisimman version. [Schultz, 2014]

Wikipedia ei luo alkuperäisistä kuvatiedostoista pikkutiedostoja (thumbnail) ennalta. Sivuille lisätään linkit pikkutiedostoihin ja pikkutiedostot generoidaan vasta, kun jokin asiakasohjelma pyytää niitä. Generoinnin jälkeen pikkutiedostot tallennetaan muiden mediatiedostojen tapaan Swiftiin. [Wikitech, 2014f] Swiftiin oli vuonna 2014 tallennettu noin 30 miljoonaa alkuperäistiedostoa sekä noin 320 miljoonaa näistä generoitua pikkukuvaa [Liambotis, 2014]

3.3.3.4 Hakutoiminto

Wikipedia käyttää sivujen indeksointiin, ja siten hakupalvelujen tuottamiseen, ElasticSearchiä [Mediawiki, 2015c]. ElasticSearchin avulla Wikipedialla on käytössään hajautettu sekä reaaliaikaisen hakupalvelu. ElasticSearchin avulla Wikipediassa on mahdollista suorittaa kokotekstihakuja (full text search).

Wikipedia tallentaa ElasticSearchiin pääosin toisistaan erilliset hakuindeksit kullekin projektilleen. Esimerkiksi englanninkieliselle Wikipedialle on oma hakuindeksinsä. Englanninkielinen Elasticsearch-hakuindeksi oli vuonna 2014 sirpaloitu kahdellekymmenelle eri palvelimelle. Kunkin palvelimen hakuindeksi oli

edelleen replikoitu kahdelle muulle palvelimelle. [Schultz, 2014] Englanninkielisen Wikipedian hakutoiminnon taustalla oli siis vuonna 2014 kaikkineen 60 Elasticsearch-instanssia.

3.3.3.5 Muuta

Wikipedia lisää tietyissä tapauksissa artikkeleihin dynaamista sisältöä. Tällaista sisältöä voi olla esimerkiksi bannerit, joilla kannustetaan käyttäjiä tekemään lahjoituksia Wikipedialle, tai ilmoitetaan ylläpitotoimista. Bannereita ei ole mahdollista lisätä suoraan artikkeleihin, koska tällöin kaikki artikkelit pitäisi poistaa välimuisteista ja kaikki kyseiset artikkelit pitäisi generoida uudelleen, jotta ne saataisiin takaisin välimuisteihin. Wikipedia käyttää ratkaisuna sivulla esiintyvää JavaScript-koodia, jonka avulla se kykenee tarvittaessa lisäämään sivulle dynaamista sisältöä. [Schultz, 2014]

Wikipedia käyttää työjonoja pitkäkestoisten töiden eräajotyyppiseen suorittamiseen. Yksi esimerkki pitkäkestoisesta työstä on mallinteen (template) päivittäminen. [Schultz, 2014] Mallinteen päivittäminen aiheuttaa sen, että myös kaikki mallinteen sisällyttävät sivut pitää päivittää. MediaWiki-ohjelmisto luo jokaista artikkelia kohden yhden eräajotyön työjonoon. [Mediawiki, 2014] Tällaisen päivityksen suorittaminen olisi hankalaa, jos se pitäisi tehdä heti sen jälkeen, kun käyttäjä on painanut mallinteen tallenna-painiketta ja käyttäjän pitäisi odottaa muutoksen tallentuvan. Käyttäjäystävällisempi sekä suorituskykyisempi tapa on siirtää tällainen raskas työ työjonoon ja prosessoida se taustaprosessointina.

MediaWiki-ohjelmisto on kykenevä lähettämään HTCP-pyynnön (**H**yper**t**ext **C**aching **P**rotocol) tilanteessa, jossa sivua päivitetään ja se on tarve poistaa Varnish-edustapalvelinten välimuisteista. [Wikitech, 2013]

3.3.3.6 Huomioita

Wikipedia näyttää eroavan esimerkiksi Twitteristä yhdellä merkittävällä tavalla. Twitterin arkkitehtuuri on selkeästi SOA-mallinen. Yksittäiset tehtävät on jaettu erillisille palveluille. Wikipedian kohdalla tilanne ei kuitenkaan ole tämä. MediaWiki-ohjelmisto on merkittävä osa koko Wikipediaa ja se muodostaa yhden suuren rakennuspalikan, jonka ympärille Wikipedia on lisännyt eri ratkaisuja suorituskyvyn ylläpitämiseksi.

Toisaalta Wikipediassakin on nähtävissä joitain SOA-arkkitehtuurin piirteitä. Tällaisia ovat esimerkiksi Parsoid- ja Mathoid-palvelut, jotka muuntavat wikitekstiä sekä matemaattisia lausekkeita HTML:ksi sekä SVG:ksi (**S**calable **V**ector

Graphics). Molemmat palvelut toimivat itsenäisesti, eivätkä pidä sisällään riippuvuutta MediaWiki-ohjelmistoon. [Wikitech, 2015, Wikitech, 2014e]

Toisaalta myös MediaWiki-ohjelmisto on hyvin selvästi mukautunut Wikipedian tarpeita vastaavaksi. MediaWiki-ohjelmisto tukee esimerkiksi isäntä-orja-kokoonpanoja sekä monia eri välimuistiratkaisuja, joiden avulla MediaWikin normaalisti synnyttämää prosessointikuormaa kyetään keventämään.

3.4 Huomioita

Kaikkia kolmea edellä kuvattua palvelua yhdistää tietyt piirteet. Tällaisia piirteitä ovat esimerkiksi käyttäjien luoma sisältö, korkea lukuoperaatioiden määrä suhteessa kirjoitusoperaatioihin sekä sen myötä kaikille yhteiseksi piirteeksi muodostuva välimuistien laajamittainen käyttö.

Twitter	Lukuoperaatiot	300000
	Kirjoitusoperaatiot	5000
Reddit	Sivujen katselukerrat	1774
	Uusien viestiketjujen lukumäärä	1,3
	Uusien kommenttien lukumäärä	12,8
	Annettujen äänien lukumäärä	212,7
Wikipedia (kaikki)	Sivujen katselukerrat	7309
	HTTP-pyynnöt	100000
Wikipedia (EN)	Sivujen katselukerrat	3515
	Uudet artikkelit	0,01
	Muokkauskerrat	1,2

Taulukko 3.1: Kirjoitus- ja lukuoperaatioiden lukumäärät eri palveluissa vuonna 2013. Lukumäärät on annettu muodossa kappaletta per sekunti.

Taulukkoon 3.1 on listattu Twitterin, Redditin ja Wikipedian osalta luku- ja kirjoitusoperaatioiden määriä. Luvut eivät ole toisensa kanssa täysin vertailukelpoisia ja samoin niiden tarkkuus vaihtelee hieman. Luvut on listattu taulukkoon, jotta edellä kuvattujen palveluiden mittasuhteita ja käyttömalleja olisi helpompi hahmottaa.

Taulukosta 3.1 näkyy muun muassa, että lukuoperaatioiden määrät ovat Twitterissä 60-kertaiset, Redditissä hieman vajaa 10-kertaiset ja englanninkielisessä

Wikipediassa peräti lähemmäs 3000-kertaiset verrattuna kirjoitusoperaatioiden määrään. Kuten edellä on kuvattu, kaikki kolme palvelua ovat täten pyrkineet optimoimaan ennen kaikkea lukuoperaatioiden suorituskykyä.

Vaikka kaikki kolme palvelua ovat keskittyneet lukuoperaatioiden suorituskyvyn optimointiin, on niiden lähestymistavoissa kuitenkin eroja. Redditille ja ennen kaikkea Twitterille on tärkeää kyetä näyttämään ajantasaista sisältöä käyttäjilleen. Kumpikin palvelu vetoaa käyttäjiinsä nimenomaan sen avulla, että niistä löytyy jokaisena vuorokauden hetkenä uutta sisältöä. Twitterissä ja Redditissä käyttäjien palveluun lisäämä sisältö menettää arvoaan ajan hampaissa hyvin nopeasti. Wikipedian kohdalla tilanne on kuitenkin toinen. Tietosanakirjassa esiintyvät artikkelit eivät menetä iän karttuessa arvoaan, monesti päin vastoin. Wikipediassa artikkelien laatu usein paranee ajan mittaan. Tämä tarkoittaa sitä, että suuri osa Wikipediaan vuosien varrella karttuneesta sisällöstä on edelleen mielenkiintoista. Wikipedian artikkelit eivät myöskään kärsi kovin siitä, mikäli niistä puuttuu esimerkiksi muutama minuutti sitten tapahtunut päivitys.

Myös käyttötavoissa ja sisällön löytämisessä on eroja. Redditissä ja Twitterissä käyttäjät seuraavat sisältöä joko toisia käyttäjiä tai aihealueita seuraamalla. Wikipedian artikkeleihin käyttäjät saapuvat useammin hakukoneilla tehtyjen hakujen tai toisista artikkeleista löytyneiden linkkien kautta.

Dynaamisuutensa vuoksi erityisesti Twitter joutuu generoimaan käyttäjien noutamat aikajanat aina uudelleen. Tämä vaade on johtanut siihen, että Twitter tallentaa välimuisteihin kaiken tarvittavan datan ja pyrkii generoimaan tästä lopputuloksen mahdollisimman nopeasti. Wikipedian kohdalla tilanne on toinen. Wikipedia kykenee suorittamaan raskaan sivun generoinnin kerran ja voi sen jälkeen jättää sivun ja sen useat välivaiheiset osat useisiin eri välimuisteihin.

Tutkimuskohteina Twitter, Reddit ja Wikipedia ovat tietyiltä osiltaan samanlaisia, tietyiltä osiltaan kovin erilaisia. Samankaltaisuuksia löytyy esimerkiksi avoimen lähdekoodin käyttämisestä sekä lähdekoodin julkaisemisesta avoimena lähdekoodina. Wikipedia käyttää toteutuksessaan ainoastaan avointa lähdekoodia ja julkaisee myös kaiken itse tekemänsä ohjelmakoodin avoimena lähdekoodina. Vastaavasti Reddit julkaisee lähes kaiken ohjelmakoodinsa avoimena lähdekoodina. Myös Twitter seuraa tätä samaa linjaa, mutta pitää monesti pidemmän viiveen, ennen kuin julkaisee lähdekoodiaan julkisesti.

Kaikki kolme esimerkkitapausta myös kuvaavat arkkitehtuureitaan ainakin jossain määrin julkisesti. Kaikista avoimmin näin tekee Wikipedia. Vaikka Wikipedia pyrkii olemaan erittäin avoin arkkitehtuurinsa suhteen, on sen tarjoamassa dokumentaatiossa silti toivomisen varaa. Selvää yhtenäistä arkkitehtuuridoku-

menttia ei Wikipedian osalta löydy, ja tietojen ajantasaisuus jää paikka paikoin arvailujen varaan. Lisäksi eri lankojen yhdistely kokonaisuuden hahmottamiseksi voi viedä enemmän aikaa kuin mitä olisi tarpeen, mikäli dokumentaatio olisi paremmin muotoiltu.

Vaikkei dokumentaatiotilanne ole Wikipedian kohdalla aivan täydellinen, on se kuitenkin merkittävästi parempi kuin Twitterin osalta.

Sekä Reddit että Wikipedia tarjoavat asennusohjeet, joiden avulla sekä Reddit-ohjelmistopaketin että MediaWikin voi pystyttää omalle palvelimelleen. Redditin ohjeet löytyvät osoitteesta: `https://github.com/reddit/reddit/wiki/Install-guide` ja MediaWikin asennusohjeet osoitteesta: `http://www.mediawiki.org/wiki/Manual:Installation_guide`.

4 Sosiaalisen median ominaispiirteitä

Tässä luvussa kerrataan ja käydään läpi piirteitä, jotka ovat ominaisia sosiaalisen median palveluille sekä oleellisia skaalautuvuuden näkökulmasta.

4.1 Datan piirteitä

4.1.1 Yleistä

Sosiaalisen median palveluun syötetylle datalle näyttää olevan tyypillistä se, että sitä lisätään suhteellisen usein, mutta pieniä määriä kerrallaan. Twitterissä 140 merkin rajoite per viesti asettaa syötettävän datan koolle selvän rajan. Redditissä monet viestiketjut aloitetaan yksittäisellä linkillä toiselle sivustolle, jonka jälkeen yksittäiset kommentit ovat useimmiten maksimissaan muutamien tekstikappaleiden mittaisia. Wikipediassa artikkeleille on tyypillistä, että ne voivat olla hieman pidempiäkin, mutta yksittäinen muokkaus ei usein ole kooltaan kovin suuri. Toisaalta kerralla lisättävän datan määrä riippuu tietenkin siitä, minkälainen palvelu on kyseessä. Edellä kuvattu ei luonnollisesti päde esimerkiksi videoidenjakopalvelu Youtubeen.

Sosiaalisen median palveluihin tallennetulle datalle on myös ominaista, ettei se ole välttämättä kovin tärkeää. Mikäli karmapiste katoaa Redditissä tai mikäli yksittäinen twiitti ei Twitterissä tallennu, ei tämä yleensä aiheuta kenellekään taloudellisia menetyksiä tai uhkaa kenenkään terveyttä. Yleensä pienet virheet tai viiveet aiheuttavat lähinnä yksittäisten käyttäjien turhautumista.

Data voidaan karkeasti luokitella kahteen kategoriaan: käyttäjien palvelun sisällä muodostamaan dataan, johon kuuluvat esimerkiksi sisältö, kommentit ja relaatiot sekä käyttäjien palveluun lisäämiin mediatiedostoihin, joihin kuuluvat muun muassa kuva-, ääni- ja videotiedostot.

Sosiaalisen median palveluille on ominaista, että käyttäjät lisäävät palveluun jatkuvalla syötöllä uutta sisältöä ja toisaalta myös se, että he odottavat pääsevänsä käsiksi muiden käyttäjien lisäämään sisältöön hyvin lyhyen ajan sisällä sisällön lisäämisen jälkeen. Jatkuvasti ja nopeasti päivittyvä sisältö on yksi niistä piirteistä, jotka tekevät sosiaalisen median palveluista sosiaalisen median palveluja.

4.1.2 Datan hakeminen

Monet sosiaalisen median palvelut tarjoavat hakupalvelun, jonka avulla on mahdollista etsiä järjestelmään tallennettua dataa. Hakupalvelujen on tarve skaalautua sekä data- että käyttäjämääriin.

Busch *et al.* [2012] kuvailevat joitain ominaisuuksia, joita Twitterin hakupalvelulta vaaditaan, mutta jotka yhtälailla pätevät muihinkin sosiaalisen median palveluihin:

- *Alhainen latenssi ja korkea suorituskyky*: Alhainen latenssi on tärkeä, koska käyttäjät odottavat saavansa hakutulokset nopeasti. Toisaalta hakupalvelun pitäisi samaan aikaan olla kykenevä suorittamaan hakuja suuresta datamassasta ja mahdollisesti palvelemaan suurta määrää samanaikaisia käyttäjiä.

- *Kyky indeksoida suuri määrä sisääntulevaa dataa sekä saattaa tämä hakupalvelun saataville nopeasti*: Esimerkiksi Twitterissä sisääntulevien uusien viestin määrä voi olla hyvinkin suuri, jopa yli satatuhatta viestiä per sekunti [Krikorian, 2013]. Tällaiset datamäärät pitää kuitenkin kyetä prosessoimaan, indeksoimaan sekä saattaa käyttäjien saataville mahdollisimman nopeasti. Sosiaalisen median palvelut keskittyvät monesti ajankohtaisiin tapahtumiin, joten datan indeksointi ei saa kestää tunteja, päiviä tai viikkoja.

- *Kyky suorittaa samanaikaisia luku- sekä kirjoitusoperaatioita*: Sosiaaliselle medialle on tyypillistä jatkuva uuden sisällön muodostuminen. Tämä tarkoittaa sitä, että hakuindeksejä on tarve kyetä päivittämään koko ajan. Hakuindeksien täytyy kuitenkin olla myös käytettävissä koko ajan. Niitä ei voi esimerkiksi lukita jatkuvasti tapahtuvien kirjoitusoperaatioiden ajaksi.

Toisaalta Busch *et al.* [2012] kuvailevat myös joitain ominaisuuksia, jotka helpottavat hakupalvelun optimointia. Toisin kuin esimerkiksi Google-haun tapaisissa WWW-sivustoja indeksoivissa palveluissa, sosiaalisen median palveluissa ei ole tarvetta miettiä, milloin jotain tiettyä sisältöä on tarve päivittää hakuindeksiin. Sosiaalisen median palvelu voi syöttää datan hakupalveluunsa sitä mukaa, kun sitä syntyy tai kun sisältöä päivitetään.

4.1.3 Negatiiviset haut

Monille sosiaalisen median palveluluille on ominaista se, että ne tarjoavat tavan pisteyttää muiden käyttäjien lisäämää sisältöä. Esimerkiksi Redditin osalta toiminta nojaa merkittävästi juuri sisällön pisteyttämisen varaan. Perinteisesti yksi käyttäjä saa antaa vain yhden pisteen kullekin palvelussa esiintyvälle sisällölle. Sosiaalisen median palvelulla on siis usein tarve kyetä tunnistamaan, onko

käyttäjä jo antanut äänensä jollekin tietylle sisällölle vai ei. Naiivi tapa tämän tiedon tallentamiseen relaatiotietokannassa olisi uuden relaation lisääminen sisällön ja käyttäjän välille, sekä uuden tietokantarivin luominen äänen antamisen yhteydessä. Paremmin tämän tiedon noutamiseen soveltuvat kuitenkin esimerkiksi Bloom-suodattimet, joiden avulla voidaan tehokkaasti varmistaa, ettei tietue kuulu tiettyyn etsittyyn joukkoon [Bradberry & Lubow, 2013, 61].

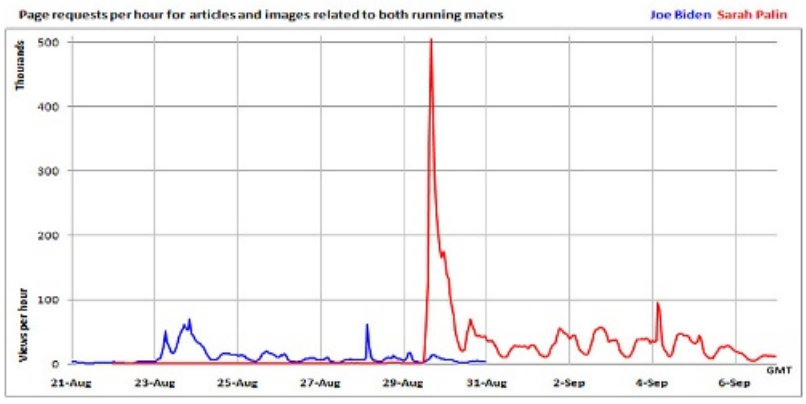
4.2 Käyttäjien piirteitä

4.2.1 Flash crowds

Yksi sosiaalisen median palveluissa esiintyvä haaste, kuten monissa muissakin WWW-pohjaisissa palveluissa, on kyky vastata tilanteisiin, joissa palvelun käyttäjämäärä kasvaa hetkellisesti merkittävästi. Ari *et al.* [2003] käyttää tällaisista tilanteista nimitystä *flash crowd*. Flash crowd on tilanne, jossa palvelun kuormitus saattaa hetkellisesti muuttua kymmen- tai jopa satakertaiseksi.

Yksi esimerkki flash crowd -tilanteesta sattui Twitterin osalta 3.8.2013, kun Japanin televisiossa esitettiin *Laputa: linna taivaalla* -animaatioelokuva. Tällöin Twitterissä julkaistujen viestien määrä nousi hetkellisesti 143199 viestiin per sekunti. Tavanomainen viestimäärä oli samana vuonna keskimäärin vain noin 5700 uutta viestiä per sekunti. [Krikorian, 2013]

Toinen esimerkki flash crowd -tilanteesta löytyy esimerkiksi Wikipediasta. Vuoden 2008 USA:n presidentinvaalien aikaan Sarah Palinia koskeva artikkeli koki flash crowd -ilmiön. Ilmiö näkyy Erik Zachten julkaisemassa kuvassa 4.1 [Wikimedia, 2015c].



Kuva 4.1 Flash crowd -tilanne englanninkielisessä Wikipediassa vuoden 2008 USA:n presidentinvaalien aikana. Kuvan on julkaissut Erik Zachte.

Myös Redditin osalta löytyy esimerkki tilanteesta, jossa palvelun kuormitus kasvoi merkittävästi. Tällainen tilanne tapahtui muun muassa vuonna 2012, kun Yhdysvaltojen presidentti Obama mahdollisti kysymysten kysymisen häneltä Redditin välityksellä. Reddit lisäsi tässä tilanteessa käyttöönsä kaikkineen 60 sovelluspalvelinta lisää. [Reddit, 2012]

4.2.2 Thundering herd

Toinen flash crowdia muistuttava tilanne muodostuu silloin, kun palvelu on ollut hetken pois toiminnasta teknisen tai muun syyn johdosta. Tässä tilanteessa käyttäjät saattavat jäädä odottamaan palvelun palaamista takaisin käyttöön. Palvelun palattua takaisin käyttöön saattavat palvelun käyttöönpalaamista odottaneet käyttäjämassat aiheuttaa palvelun hetkittäisen ylikuormittumisen ja siten ajaa palvelun takaisin tilaan, jossa se ei ole enää käytettävissä. Erdberg käyttää termiä *thundering herd* tilanteesta, jossa palvelu palaa takaisin käyttöön ja käyttäjät rynnivät käyttämään sitä [Edberg, 2013a].

4.3 Muita huomioita

Tutkielmassa käsitellyille sosiaalisen median palveluille on ominaista, että niiden käyttäjäkunta on globaalia. Tämä tarkoittaa sitä, että palvelun tulee olla aina toiminnassa, ja ettei varsinaisia huoltoikkunoita palvelun alas ajamiselle ja päivittämiselle ole.

5 Skaalautuvuuden ratkaisumalleja

5.1 Yleistä

Tässä tutkielmassa on määritelty sosiaaliselle medialle ominaiseksi piirteeksi se, että sosiaalisessa mediassa käyttäjät sekä luovat että kuluttavat sisältöä. Tutkielmassa on keskitytty sosiaalisen median palveluihin, joilla on suuria käyttäjämääriä. Suuret käyttäjämäärät yhdistettynä näiden käyttäjien jatkuvasti palveluun lisäämään uuteen sisältöön muodostavat sen arkkitehtuurillisen haasteen, johon tämä tutkielma pyrkii etsimään vastauksia.

Tässä luvussa syvennytään kuvaamaan arkkitehtuurillisia malleja, joita on käytetty muun muassa luvussa 3 kuvatuissa sosiaalisen median palveluissa. Nämä mallit mahdollistavat skaalautuvuuden sosiaalisen median kehyksessä, mutta kuvatut mallit eivät sinällään ole uusia tai sosiaaliseen median kehykseen sidottuja.

Skaalautuvuuden perustana näyttää pohjimmiltaan olevan kyky pitää yksittäiseen palvelimeen kohdistuva rasitus kohtuullisena, järjestelmään kohdistuvasta kuormasta riippumatta. Koska järjestelmään kohdistuvaa kuormitusta ei ole mahdollista rajata, ja koska yksittäisen palvelimen tehoja ei ole mahdollista kasvattaa määräänsä enempää (vertikaalinen skaalautuvuus), näyttää ratkaisukeinoksi jäävän horisontaalinen skaalautuvuus eli kyky hajauttaa palvelun toiminnoista aiheutuva kuormitus useille eri palvelimille.

Sosiaalisen median palvelujen käsittelemät datamäärät voidaan mieltää big data -kategorian alle. Big datalla tarkoitetaan datamääriä, jotka ovat niin suuria, että niitä on hankala tallentaa, hallinnoida ja analysoida perinteisillä tietokantaratkaisuilla. [Hansmann & Niemeyer, 2014]

Palvelulta vaadittavien piirteiden suhteen tavoitteet voidaan kiteyttää vastaamaan Gualtierin [2012] esittämiä tavoitteita, joita kaivataan järjestelmiltä, jotka käsittelevät big dataa:

- järjestelmän kyky tallentaa data
- järjestelmän kyky prosessoida data
- järjestelmän kyky noutaa ja suorittaa hakuja tallennettuun dataan.

Tutkielmassa läpikäytyjen esimerkkitapausten, Twitterin, Redditin ja Wikipedian, pohjalta on havaittavissa ainakin seuraavat piirteet, joiden avulla palvelut pyrkivät skaalautumaan:

- tekniikat kuormituksen hajauttamiseksi ja vähentämiseksi
- laskennan siirtäminen myöhemmäksi
- tietokanta- ja tallennusratkaisujen monimuotoisuus
- datan sirpalointi
- datan replikointi.

Tässä luvussa syvennytään käsittelemään niitä tekniikoita, joiden avulla järjestelmässä tapahtuvaa prosessointia sekä datan tallentamista kyetään sekä hajauttamaan että tehostamaan.

5.2 Palvelupyynnön käsittely

WWW-pohjaisissa palveluissa, kuten Twitterissä, Redditissä ja Wikipediassa, järjestelmään kohdistuva kuorma muodostuu suurelta osin HTTP-palvelupyynnöistä. Palvelun käyttäjä avaa palvelun sivun selaimessa ja tästä muodostuu joukko pyyntöjä, joiden avulla selain noutaa sivun sisällön sekä sivuun liittyvät staattiset resurssit, kuten kuva-, tyyli- ja JavaScript-tiedostot.

Luokitellaan selaimelta tai muulta asiakasohjelmalta palveluun kohdistuvat pyynnöt seuraavasti: karkealla tasolla pyynnöt jakaantuvat luku- ja kirjoituspyyntöihin, joista edelleen muodostuu palveluun luku- ja kirjoitusoperaatioita. Lukupyynnöt jakaantuvat edelleen pyyntöihin, joilla noudetaan sisältöä, staattisia resursseja, kuten tyyli- ja JavaScript-tiedostoja tai mediatiedostoja, kuten kuva- ja äänitiedostoja. Mediatiedostot voivat olla palvelun ulkoasuun kuuluvia tiedostoja tai käyttäjien palveluun lisäämää sisältöä. Suorien lukuoperaatioiden ohella palveluun voi kohdistua myös hakupyyntöjä. Hakupyyntöjen pohjalta palvelu etsii käyttäjän haluaman sisällön ennen lukuoperaation suorittamista. Lukuoperaation lopputuloksena palvelu palauttaa selaimelle vastauksen, joka sisältää pyydetyn resurssin.

Edelleen kirjoituspyynnöt voidaan jakaa muutamaan eri vaihtoehtoon. Kirjoituspyyntöjen avulla palveluun voidaan lisätä sisältöä, päivittää vanhaa sisältöä, mahdollisesti poistaa sisältöä sekä esimerkiksi päivittää käyttäjätiliin tai käyttäjän sosiaaliseen verkostoon liittyviä tietoja. Kirjoitusoperaation myötä palveluun voidaan tallentaa tekstimuotoista sisältöä tai mediatiedostoja. Kirjoitusoperaatio

johtaa monissa tapauksissa palvelun sisällä eri operaatioiden suorittamiseen, kuten datan tallentamiseen, hakuindeksien päivittämiseen ja välimuistien päivittämiseen. Kirjoitusoperaatio voi johtaa myös eräajotyylisten tehtävien liipaistumiseen ja suorittamiseen. Lisäksi esimerkiksi Twitterissä uuden viestin tallentamisen myötä Twitterille muodostuu tarve edelleen välittää kyseinen viesti WWW-sovelluspalvelujensa välityksellä muille, palvelun ulkopuolisille, osapuolille.

HTTP-pyynnöt käsitellään toisistaan riippumattomasti. Pyynnön käsittelytapa riippuu pyynnön laadusta. Esimerkiksi Reddit tarjoilee lukuoperaatioiden osalta sisältönsä Akamain sisällönjakelupalvelua hyödyntäen, mikäli pyynnön suorittaa sisäänkirjautumaton käyttäjä. Sisäänkirjautuneiden käyttäjien kohdalla Reddit haarauttaa pyynnön käsittelyn sen mukaan, ollaanko noutamassa staattisia resursseja vai dynaamista sisältöä. Staattisten resurssien osalta pyynnöt ohjautuvat sellaisenaan Amazon pilvipalveluvalikoimasta löytyvälle Simple Storage Servicelle (S3). Dynaaminen sisältö muodostetaan Redditin omien sovelluspalvelinten avulla.

Wikipedia toimii lukupyynnön käsittelyn osalta samankaltaisesti, pyrkien ensisijaisesti tarjoamaan sisällön omilta Varnish-edustapalvelimiltaan. Mikäli sisältöä ei näiden palvelinten välimuisteista löydy, jakaantuu pyyntö joko sovelluspalvelinten suoritettavaksi tai mikäli pyyntö koskee mediatiedostoja, ohjautuu pyyntö taustalta löytyvälle Swift-mediatiedostovarastolle.

Sisältöä palvelusta noudettaessa pyyntö ohjautuu ensi vaiheessa joko suoraan palvelun hyödyntämille välimuisteille tai kuormantasauksen kautta sovelluspalvelimille ja sieltä edelleen eri välimuisti- ja tietokantapalvelimille, sekä muille palvelun taustalta löytyville palvelimille.

5.3 Kuormantasaus

Ensimmäinen vaihe pyynnön käsittelyssä on pyynnön vastaanottaminen ja päätöksen tekeminen siitä, missä ja millä tavoin pyyntö olisi parasta käsitellä. Pyynnöstä riippuen vastaus pyyntöön voidaan noutaa joko välimuistipalvelimilta (edustapalvelimet tai sisällönjakelupalvelimet) tai sovelluspalvelimilta. Ennen pyynnön käsittelyä on kuitenkin kyettävä päättämään, mille palvelussa esiintyvällä palvelinresurssille pyyntö olisi parasta ohjata. Yksinkertaisimmassa tapauksessa pyynnöt pyritään ohjaamaan palvelinresursseille esimerkiksi kiertovuorottelualgoritmia (round robin) hyödyntäen siten, että palveluun kohdistuva kuormitus jakaantuu palvelimille tasaisesti. Monimutkaisemmissa tapauksissa pyynnön ohjaukseen voi kuitenkin vaikuttaa myös muut tekijät, kuten esimerkiksi tietover-

kossa olevat hetkittäiset viiveet tai jopa sähkön sen hetkinen hinta [Busch *et al.*, 2012].

Palvelun ollessa globaali ja käyttäjämäärältään riittävän suuri, palvelun toiminnot hajautetaan usein maantieteellisesti toisistaan erillään oleviin palvelinkeskuksiin. Palvelun toiminnot hajauttamalla on mahdollista saavuttaa etuja esimerkiksi luotettavuuden, kustannusten sekä käyttäjille muodostuvien, maantieteellisestä sijainnista aiheutuvien, viiveiden suhteen. Ensimmäinen askel kuormantasauksessa on tällaisessa tapauksessa sopivimman palvelinkeskuksen valitseminen. Esimerkiksi Wikipedian kohdalla tämä askel on toteutettu Wikipedian itsensä ylläpitämien nimipalvelimien avulla. Näiden nimipalvelinten avulla nimikyselypyynnöille muodostetaan vastaukset, joiden avulla asiakasohjelmistot ohjataan maantieteellisesti lähimmälle palvelinkeskukselle.

Palvelinkeskuksessa pyynnön käsittelyyn osallistuu edelleen joukko eri resursseja. Pyyntöön saatetaan kyetä muodostamaan vastaus edustapalvelinten välimuistien avulla, tai pyyntö saatetaan joutua ohjaamaan taustalta löytyville sovellus- ja muille palvelimille.

Pyyntöä ensimmäistä kertaa sovelluspalvelimille ohjatessa, voidaan pyynnön käsittelyyn yleensä valita mikä tahansa käytettävissä oleva sovelluspalvelin. Algoritmi palvelimen valintaan voi olla tällöin hyvin yksinkertainen. Sekä Reddit että Wikipedia hyödyntävät sovelluspalvelimen valintaan kiertovuorottelualgoritmia. Kiertovuorottelualgoritmissa palvelimet on listattu listaan ja pyynnöt ohjataan järjestyksessä aina edellistä seuraavalle palvelimelle. Listan viimeisen palvelimen jälkeen siirrytään jälleen listan alkuun.

Mikäli sovelluspalvelimella ylläpidetään tilatietoa käyttäjän istunnosta, voi olla tarpeellista, että kaikki käyttäjän pyynnöt saadaan ohjattua samalle palvelimelle. Redditin käyttämä HAProxy ratkaisee tämän lisäämällä vastaukseen evästeen, josta löytyy käytetyn sovelluspalvelimen tunniste (sticky session). Selain lähettää evästeen automaattisesti myöhempien pyyntöjen mukana, joten HAProxy kykenee ohjaamaan yksittäisen käyttäjän pyynnön aina yhdelle ja samalle palvelimelle.

Tutkielmassa tutkituista palveluista Reddit käyttää kuormantasaukseen HAProxy-ohjelmistoa ja Wikipedia LVS-DR:ää, jonka toimintaperiaate on kuvattu luvussa 3.3.3. Twitterin osalta tarkkoja tietoja kuormantasauksesta ei ole, vaikka Twitterinkin osalta löytyy viitteitä HAProxyn käytöstä [HAProxy, 2015].

5.4 Välimuistien käyttäminen

Kaikki kolme tutkittua esimerkkitapausta hyödyntävät eri tasoisia välimuisteja pienentääkseen sovellus- ja tietokantapalvelimille kohdistuvaa kuormitusta. Välimuistin tavoitteeksi muodostuu kyky pitää käsillä sellaista dataa, jota tarvitaan usein, tai jonka noutaminen tai muodostaminen on järjestelmälle kallista.

Esimerkiksi Twitterin osalta Aniszczyk [2012] tiivistää Twitterin tarpeet välimuisteille kahteen kategoriaan:

1. Muistinvarassa toimivien välimuistien käytöllä Twitter voi poistaa kuormaa tietokannoilta ja nopeuttaa datan noutoa verrattuna tilanteeseen, jossa data noudettaisiin kiintolevyiltä.

2. Välimuistien käyttö mahdollistaa prosessointikuormituksen vähentämisen sallimalla sellaisten objektien väliaikaisen tallentamisen, joiden muodostaminen on laskennallisesti kallista.

Välimuisteihin tallennettu data voi olla tietokannasta noukittua dataa tai se voi olla järjestelmässä suoritetun laskennan erivaiheisia lopputuloksia. Esimerkiksi Wikipedia tallentaa Varnish-välimuisteihin kokonaisia HTML-muotoisia sivuja sekä mediatiedostoja. Pinnan alla Wikipedia käyttää MediaWiki-ohjelmiston piirteitä hyväkseen ja käyttää Memcachedia useiden MediaWiki-ohjelmiston tarvitsemien objektien tallentamiseen.

Vaikka välimuistien ensisijainen tehtävä on kuorman vähentäminen, voidaan välimuistien käytöllä saavuttaa myös muita etuja. Välimuistien avulla voidaan esimerkiksi mahdollistaa palvelun käytettävyys tilanteessa, jossa järjestelmän sovellus- tai tietokantapalvelimiin kohdistuu virhe- tai päivitystilanne. Esimerkiksi Reddit kykenee tarjoamaan staattista sisältöä käyttäjien luettavaksi Akamain sisällönjakelupalvelun sekä Memcachedin avulla, vaikka jokin osa Redditin muusta järjestelmästä olisikin virhetilassa [Edberg, 2013a].

Välimuistit voidaan edelleen luokitella sen mukaan, lasketaanko niihin arvot ennalta vai tallentavatko ne prosessoinnin lopputuloksen vasta pyynnön suorittamisen jälkeen. Wikipedia näyttää suosivan artikkelien sekä laskennan välitulosten tallentamista vasta siinä vaiheessa, kun niille on tarvetta. Twitter ja Reddit suosivat välimuistien osittaista täyttämistä jo ennalta. Tämä selittynee osin palvelujen eriluonteisilla käyttötavoilla. Esimerkiksi Redditissä suosituimmat osiot ovat ennalta hyvin tiedossa, ja siten tätä tietoa voidaan helposti käyttää hyväk-

si palvelun suorituskyvyn kohentamiseen. Vastaavaan tapaan Twitter ylläpitää välimuisteissa ainoastaan aktiivisiksi katsomiensa käyttäjien aikajanat.

5.4.1 Hajautetut välimuistit

Välimuistien osalta kohdataan sama haaste kuin muidenkin järjestelmään kuuluvien palvelinten osalta. Yksittäiselle välimuistipalvelimelle tallennetun datan määrä ja siten siihen kohdistuva rasitus olisi kyettävä pitämään kohtuullisena. Lisäksi yksittäiselle välimuistipalvelimelle ei usein ole mahdollista tallentaa kaikkea sitä dataa, joka välimuisteihin tahdottaisiin säilöä.

Palvelun tulee siis kyetä tallentamaan eri välimuistipalvelimille toisistaan erilliset osajoukot dataa. Tästä seuraa se, että pyyntöä vastaanotettaessa pyyntö tulee kyetä ohjaamaan sille välimuistipalvelimelle, jolta etsitty data todennäköisimmin löytyy. Kuormantasaus välimuistipalvelimille vaatii siis monimutkaisempaa lähestymistapaa kuin esimerkiksi sovelluspalvelimille suoritettava kuormantasaus.

Hajautettujen välimuistien (distributed cache) toteuttamiseen on esitetty useita eri tekniikoita. Malpani *et al.* [1995] esittivät, että välimuistipalvelimille kohdistuvat pyynnöt voitaisiin jakaa sattumanvaraisesti eri palvelimille. Mikäli haettua tietuetta ei löytyisi tältä sattumanvaraisesti valitulta palvelimelta, lähettäisi kyseinen palvelin *ryhmälähetyksenä* (multicast) kyselyn kaikille muille järjestelmän välimuistipalvelimille. Mikäli jokin välimuistipalvelin vastaisi pyyntöön sopivan aikaikkunan sisällä, noudettaisiin tietue sieltä. Jos vastausta ei saataisi, noudettaisiin tietue sovellus- tai tietokantapalvelimilta ja tallennettaisiin välimuistiin, jotta se löytyisi sieltä seuraavilla hakukerroilla. Muun muassa Karger *et al.* [1997] toteavat kuitenkin, että ratkaisu on huono, koska välimuistipalvelinten määrän kasvaessa ryhmälähetysviestien määrä kasvaa liian suureksi.

Monet esitetyt ratkaisut hajautettujen välimuistien toteuttamiseen tekevät oletuksen, että eri välimuistien on tarve tavalla tai toiselle tietää, mitä tietueita muut rinnakkaiset välimuistit sisältävät. Intuitiivisesti tämä tuntuu oikealta päätelmältä.

Chankhunthod *et al.* [1996] esittävät ratkaisun, jossa välimuistit muodostavat puurakenteen. Pyyntö kohdistetaan puun lehtisolmuun. Mikäli lehtisolmulla on haettu tietue, palauttaa se sen suoraan. Jos lehtisolmulla ei ole haettua tietuetta, kysyy lehtisolmu tietuetta sisariltaan sekä vanhemmaltaan. Pyyntö jatkaa puussa rekursiivisesti eteenpäin, kunnes puun juurisolmu saavutetaan ja pyyntö ohjataan tarpeen vaatiessa tietueen alkuperäiselle lähteelle.

Gadde *et al.* [1997] esittävät ratkaisun, jossa tieto eri välimuistipalvelimien si-

sältämistä tietueista tallennettaisiin keskitettyyn hakemistoon. Välimuistiin kohdistuva pyyntö voitaisiin tässä tapauksessa ohjata mille tahansa välimuistipalvelimelle. Mikäli kyseiseltä palvelimelta ei löytyisi haettua tietuetta, voisi kyseinen palvelin kysyä keskitetyltä hakemistolta, mistä tietue löytyy, ja noutaa sen hakemiston osoittamasta osoitteesta.

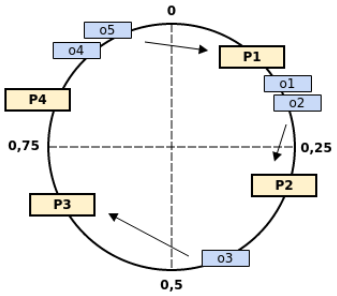
Edelleen Fan *et al.* [2000] ehdottavat toteutusta, jossa kultakin välimuistipalvelimelta löytyisi yhteenveto tietueista, jotka muilta välimuistipalvelimilta löytyvät. Tässä ratkaisussa ei olisi tarvetta lähettää ryhmälähetyskyselyjä tai ylläpitää keskitettyä hakemistopalvelinta. Toisaalta välimuistipalvelimien olisi silti edelleen tarpeen kommunikoida keskenään, mitä tietueita miltäkin palvelimelta löytyy.

Paras tilanne olisi kuitenkin se, jossa pyyntö kyettäisiin ohjaamaan oikealle välimuistille ilman, että välimuistien tarvitsee ylläpitää tietoa toisistaan. Yhden mahdollisuuden tähän tarjoaa esimerkiksi HAProxy, joka osaa laskea pyydetystä URL-osoitteesta hajautusarvon ja tarjoaa mahdollisuuden ohjata pyyntö tämän hajautusarvon pohjalta hajautusarvoa vastaavalle palvelimelle. Oikea palvelin löydetään jakamalla hajautusarvo palvelinten painoarvoilla ja ottamalla lopputuloksesta jakojäännös [Tarreau, 2015]. Algoritmi on muotoa $hash(URL) \bmod w$, jossa hajautusarvofunktio *hash* muuntaa merkkijonon numeeriseksi arvoksi.

Edellä kuvattu naiivi algoritmi aiheuttaa kuitenkin ongelmia, mikäli järjestelmään on tarve lisätä uusia välimuistipalvelimia tai poistaa vanhoja. Palvelimen poistaminen tai uuden lisääminen tarkoittaa edellä kuvatun algoritmin tapauksessa jakajan arvon muuttumista, ja siten sitä, että kunkin välimuistiin tallennetun tietueen osalta pitäisi laskea uudelleen, mille palvelimelle tietue kuuluu ja tarvittaessa siirtää resurssi kyseiselle palvelimelle.

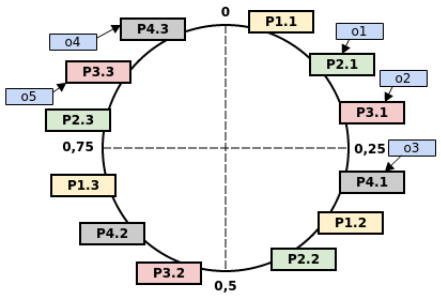
Karger *et al.* [1997, 1999] ehdottivat 1990-luvun lopulla ratkaisuksi *johdonmukaisten hajautusarvojen* (consistent hashing) käyttöä. Heidän ehdottamassaan toteutuksessa sekä URL-merkkijonoille että välimuistipalvelimille lasketaan hajautusarvot. Hajautusarvofunktion tulee muuntaa merkkijonot sekä välimuistipalvelinten tunnisteet numeerisiksi arvoiksi, jotka ovat välillä $[0, 1]$. Karger *et al.* [1999] kuvaa, että näin saadut numeeriset arvot voidaan mieltää yksikköympyrän arvoiksi. Karger *et al.* [1999] kuvaa edelleen, että palvelimet voidaan asettaa tämän lasketun arvon pohjalta yksikköympyrän kehälle. Välimuistiin tallennettaville tietueille lasketaan vastaavaan tapaan hajautusarvot ja tietueet sijoitetaan yksikköympyrän kehälle. Tämän jälkeen tietueet tallennetaan lähimmälle välimuistipalvelimelle, joka löytyy yksikköympyrän kehää myötäpäivään kiertäen kuvan 5.1 esittämällä tavalla.

Johdonmukaisia hajautusarvoja käytettäessä kunkin palvelimen tallennustila



Kuva 5.1 Objektien o1, o2, o3, o4 ja o5 jakautuminen palvelimille P1, P2, P3 ja P4.

voidaan edelleen jakaa useampaan osaan. Karger *et al.* [1999] esitti alkuperäisessä mallissaan, että kukin palvelin osoitetaan yksikköympyrän kaarelle laskemalla palvelimen tunnisteesta hajautusarvo. Tätä mallia on edelleen mahdollista laajentaa jakamalla kunkin palvelimen käytettävissä oleva tallennustila pienempiin lohkoihin (virtual node) ja antamalla kullekin lohkolle oman tunnisteensa. Jokainen lohko voidaan edelleen sijoittaa erikseen yksikköympyrän kehälle käyttämällä esimerkiksi kaavaa $hash(palvelimen\_tunniste + lohkon\_tunniste)$. Tästä syntyvää lopputulosta on hahmoteltu kuvassa 5.2.



Kuva 5.2 Objektien o1, o2, o3, o4 ja o5 jakautuminen palvelimille P1, P2, P3 ja P4. Kunkin palvelimen tallennustila on jaettu kolmeen virtuaaliseen lohkoon.

Edellä kuvattua tallennustilan lohkoihin jakamista käytetään esimerkiksi Amazonin pilvipalveluvalikoimasta löytyvässä DynamoDB:ssä [DeCandia *et al.*, 2007] sekä Memcachedin ja Redisin kaltaisessa avain-arvo-tietokanta Riak:ssa [Riak, 2015b]. DeCandia *et al.* [2007] kuvailevat tämänkaltaisen lohkomisen eduiksi sitä, että palvelinta poistettaessa data jakaantuu tasaisemmin poistettavalta palvelimelta muille palvelimille ja vastaavasti uutta palvelinta lisättäessä uusi pal-

velin pienentää tasaisemmin muiden palvelinten kuormaa. Lisäksi DeCandia *et al.* [2007] huomattavat, että lohkomisella mahdollistetaan myös toisistaan suorituskyvyltään eroavien palvelinten käyttäminen. Mikäli palvelin on tehokkaampi, voidaan sen tallennustila jakaa useampaan lohkoon ja se saa siten automaattisesti suuremman kuorman vastuulleen.

Tutkituista esimerkkitapauksista kaikki näyttävät päätyneen hyödyntämään välimuistitoteutuksissaan johdonmukaisia hajautusarvoja tavalla tai toisella. Esimerkiksi Wikipedian kohdalla pyynnöt jaetaan Varnish-välimuistipalvelimille kahden toisistaan erillisen askeleen kautta. Ensimmäisen askeleen myötä pyyntö välitetään LVS-DR-kuormantasauksen avulla kiertovuorottelualgoritmilla Varnish frontend -palvelimelle, jonka ainoana tehtävänä on laskea pyynnöstä muodostuva hajautusarvo ja siten välittää pyyntö oikealle Varnish backend -palvelimelle, josta haetun sisällön oletetaan löytyvän.

Tutkituissa tapauksissa on myös ajan saatossa esiintynyt myös joitain kömmähdyksiä välimuistitoteutusvalintojen suhteen. Erityisesti Redditin arkkitehtuuri näyttää alkuvaiheilla kompastelleen välimuistitoteutuksensa kanssa. Huffman ja Williams [2014] kuvailevat, miten palvelun alkuaikoina jokaisella sovelluspalvelimella oli paikallinen (local) välimuisti. Sovelluspalvelimia oli useita, ja jokainen näistä palvelimista synkronoi tietonsa kaikkien muiden sovelluspalvelinten kanssa Spread-ohjelmistoa [Spread, 2015] käyttäen. Tämä ratkaisu ei luonnollisesti ollut skaalautuva, koska jokaisen uuden palvelimen lisääminen kasvatti verkkoliikenteen määrää.

5.4.2 SSD-kiintolevyt keskusmuistin jatkeena

Keskusmuistin varassa toimivaa välimuistia on mahdollista laajentaa ottamalla keskusmuistin ohelle käyttöön SSD-pohjaisia kiintolevyjä (**S**olid **S**tate **D**isk hard drive). Kiintolevyt ovat kautta historian tarjonneet suuremman tallennuskapasiteetin, pienemmin kustannuksin kuin tavallinen keskusmuisti. Perinteiset pyöriviin magneettilevyihin pohjautuvat kiintolevyt ovat kuitenkin aina olleet selvästi keskusmuistia hitaampia muun muassa datan haku- ja siirtoaikojen suhteen. SSD-kiintolevyt muuttavat kuitenkin tätä asetelmaa. SSD-kiintolevyt sallivat sekä hyvin pienillä viiveillä suoritettavat datan haut että kyvyn nopeisiin luku- ja kirjoitusoperaatioihin. Muun muassa Edberg kuvailee, että SSD-kiintolevyt olisi parempi mieltää hitaaksi keskusmuistiksi kuin tavallisten perinteisten kiintolevyjen korvaajiksi [Edberg, 2013b].

Tätä näkemystä tukee myös Twitterin käytössä ollut (mahdollisesti edelleen

käytössä oleva) ja avoimena lähdekoodina julkaistu *Fatcache*-ohjelmisto. Fatcachen dokumentaatio [Fatcache, 2013] kuvailee, että vaikka SSD-kiintolevyjen suorituskyky on selvästi keskusmuistia heikompi, tarjoavat ne selkeitä etuja kokonsa, hintansa sekä muun muassa sähkönkulutuksensa puolesta. Fatcachen dokumentaatio kuvailee, että mikäli käytettävä välimuisti on verkkoyhteyden päässä, aiheuttaa verkkoyhteys usein SSD-kiintolevyjen suorituskykyä suuremman pullonkaulan. Tällaisessa tilanteessa erot keskusmuistiin tallentamisen ja SSD-kiintolevylle tallentamisen välillä kääntyvät SSD-kiintolevylle tallentamisen puolelle. [Fatcache, 2013]

Välimuistikäytön lisäksi SSD-kiintolevyjen avulla on mahdollista parantaa myös tavallisten tietokantapalvelimien suorituskykyä. Edberg kuvailee SSD-kiintolevyjen käyttöönoton parantaneen merkittävästi Redditin käytössä olleen PostgreSQL-tietokannan suorituskykyä. [Edberg, 2013b]

5.5 Relaatiotietokannat

Relaatiotietokannat ovat olleet suosittuja välineitä datan tallentamiseen aina 1990-luvulta lähtien. Relaatiotietokannat ovat tuttuja kehittäjille, ja sekä niiden hyvät että huonot puolet tunnetaan hyvin. Kaikki kolme tutkittua esimerkkitapausta käyttävät relaatiotietokantoja hyväkseen. Tutkitaan seuraavaksi, minkälaisia piirteitä relaatiotietokannat tarjoavat, ja miten ne vastaavat palvelujen kaipaamiin skaalautuvuustarpeisiin.

Relaatiotietokantojen yhteydessä puhutaan transaktioista (transaction). Transaktioilla tarkoitetaan relaatiotietokantojen kykyä niputtaa useampi luku- ja/tai kirjoitusoperaatio yhdeksi nipuksi, ja käsitellä tätä nippua atomisena kokonaisuutena vastaavasti kuin käsiteltäisiin yhtä yksittäistä operaatiota. Relaatiotietokantojen kykyä käsitellä transaktioita atomisesti ja siten pitää tietokanta eheänä kuvataan kirjainlyhenteellä *ACID*, joka muodostuu sanoista atomisuus (**A**tomicity), eheys (**C**onsistency), eristyneisyys (**I**solation) sekä pysyvyys (**D**urability). ACID-piirteet on kuvattu useissa lähteissä. Elmasri ja Navathe [2007, 606] kuvaavat ne seuraavasti:

- *Atomisuus*: atomisuudella viitataan siihen, että järjestelmässä suoritettu transaktio joko tapahtuu kokonaan tai on kokonaan tapahtumatta. Järjestelmä siirtyy aina yhdestä *eheästä* tilasta toiseen, tai on kokonaan siirtymättä.

- *Eheys*: eheydellä tarkoitetaan esimerkiksi vierasavaimien ja muiden tietokantaan asetettujen rajoitteiden määrittämää eheyttä, ja tietokannan kykyä pitää tilansa aina eheänä.

- *Eristyneisyys*: eristyneisyydellä viitataan siihen, ettei transaktion suorittaminen saa häiritä muita samaan aikaan järjestelmässä suoritettavia transaktioita.

- *Pysyvyys*: pysyvyys tarkoittaa sitä, että onnistuneesti suoritetun transaktion jälkeen data on kirjoitettu järjestelmään pysyvästi, ja se ei saa sieltä enää kadota esimerkiksi virran katkeamisen tai muun vastaavan virhetilanteen johdosta.

Jotta ACID-piirteet kyettäisiin takaamaan, on tietokantamoottorin käytännössä kyettävä rajoittamaan samanaikaisuutta. Yleisesti käytetty tapa on datan lukitseminen transaktion keston ajaksi siten, etteivät muut transaktiot kykene joko lukemaan tai päivittämään kyseistä dataa samanaikaisesti. Lukitseminen aiheuttaa usein sen, että eri transaktiot päätyvät odottamaan toistensa lukitsemia resursseja. Tällaisessa tilanteessa transaktioiden suoritus voi myös päätyä umpikujaan (deadlock). Toistensa kanssa umpikujaan päätyneistä transaktioista osa täytyy perua, jotta muut transaktiot kyettäisiin suorittamaan loppuun. Perutut transaktiot voidaan suorittaa tämän jälkeen uudelleen. Lukkojen luonteesta johtuen niitä käytettäessä tietokannan suorituskyky usein hiipuu sitä enemmän, mitä suurempi kuorma siihen kohdistuu. Skaalautuvuuden näkökulmasta tämä ei luonnollisesti ole hyvä piirre.

Vaikka ACID-piirteet ovat hyödyllisiä datan eheyden ja oikeellisuuden varmistamisen kannalta, aiheuttavat ne samalla ongelmia suorituskyvyn sekä skaalautuvuuden suhteen. Yksinkertaisimmin ACID-piirteet kyetään säilyttämään, mikäli koko tietokanta sijaitsee yhdellä palvelimella.

Mikäli relaatiotietokanta on tahdottu pitää yhdellä palvelimella, on vaihtoehdoiksi suorituskyvyn kohentamiseksi pääosin jääneet palvelimen suorituskyvyn kohentaminen sekä tehokkaampien levyjärjestelmien käyttäminen. Lisäksi tietokannan dataa on voitu *osittaa* (partition) levyjärjestelmään siten, että yhden ison tietokantatiedoston sijaan tietokantaan tallennettua dataa on pilkottu useisiin pienempiin tiedostoihin, jotka on edelleen voitu sijoittaa toisistaan erillisille fyysisille massamuisteille. Vaikka näin on kyetty rakentamaan suorituskykyisiä ratkaisuja, ei kyseinen malli kuitenkaan tue hyvin esimerkiksi sosiaalisen median palveluille ominaista jatkuvaa käyttäjä- ja datamäärien kasvua.

Mikäli kuormaa on haluttu jakaa useammalle tietokantapalvelimelle, on vaihtoehtoina olleet muun muassa erilaiset rypäsjärjestelmät (cluster system). Rypäsjärjestelmissä usean palvelimen kokoonpano toimii yhdessä näyttäen käyttäjälle yhdeltä järjestelmältä. Ryppään tavoitteena on tarjota sen käyttäjälle vastaavat piirteet, jotka käyttäjä saisi yksittäistä tietokanta käyttäessä, mutta paremmalla suorituskyvyllä. Rypäsjärjestelmät luokitellaan sen mukaan, toimivatko palvelimet täysin toisistaan erillään (shared nothing architecture) [Stonebraker, 1986], onko niiden kesken jaettu massamuistia (shared disk architecture) tai onko niiden kesken jaettu keskusmuistia (shared memory architecture) [Elmasri & Navathe, 2007, 859].

ACID-piirteet kyetään rypäsjärjestelmissä toteuttamaan hajautettuja transaktioita (distributed transaction) käyttäen. Yhden vaihtoehdon hajautettujen transaktioiden toteuttamiseen tarjoaa two-phase commit -protokolla (2PC protocol). 2PC-protokollassa transaktion eteenpäin viemisestä huolehtii erillinen koordinaattori (coordinator), joka varmistaa, että transaktio saadaan joko kokonaan suoritettua kaikilla siihen osallistuvilla palvelimilla, tai virhetilanteen sattuessa kokonaan peruttua [Elmasri & Navathe, 2007, 674]. Hajautetun transaktion suorittamiseen on ehdotettu myös muita ratkaisuja, kuten Paxos-algoritmia, jossa ei ole tarvetta keskitetylle koordinaattorille [Lamport, 2001, Gray & Lamport, 2006]. Tutkittujen esimerkkitapausten osalta ei näy viitteitä rypästietokantajärjestelmien käytöstä.

Käytännössä suorituskyky muodostuu usein haasteeksi relaatiotietokantojen yhteydessä, ja usein suorituskykyä on pyritty kasvattamaan ACID-piirteiden kustannuksella. Muun muassa Sadalage ja Fowler [2012, 53] huomauttavat, että esimerkiksi transaktioiden eristyneisyystasoa on ollut usein tapana tiputtaa perinteisissäkin järjestelmissä suorituskyvyn kohentamiseksi. Näin toimimalla menetetään kuitenkin osittain ACID-piirteisiin kuuluva eristyneisyys.

Muita tapoja suorituskyvyn kohentamiseen on ollut muun muassa transaktioiden tarjoaman turvallisuuden korvaaminen kompensaatiolla (compensation). Kompensaatiossa virheet tai epäyhtenevyydet datassa sallitaan joksikin aikaa, ja ne pyritään huomaamaan sekä korjaamaan myöhemmin.

Toinen yleisesti relaatiotietokantojen yhteydessä käytetty optimointikeino on denormalisaatio (de-normalization), jossa toisistaan erillistä, mutta kuitenkin toisiinsa liittyvää dataa ei eroteta erillisiin tietokantatauluihin. Denormalisoinnin avulla on mahdollista vähentää JOIN-lauseiden käsittelystä aiheutuvaa kuormaa. Sekä JOIN-lauseiden välttely että denormalisaatio sotivat kuitenkin omalla tavallaan tarkoitustaan vastaan, koska nimensä mukaisesti relaatiotietokantojen kan-

tava ajatus on datassa esiintyvien relaatioiden kuvaaminen.

Muihin optimointitapoihin kuuluu muun muassa tavoite suorittaa kyselyt pääasiassa indeksejä käyttäen. Muun muassa Wikipedia kuvaa ohjeistuksessaan, että kaikki tietokantakyselyt tulisi kyetä suorittamaan indeksejä hyväksikäyttäen. Toisaalta vastapainona jokainen indeksi vaatii sekä levytilaa että prosessointitehoa datan päivitysten yhteydessä. Indeksien määrän kasvaessa riittävän suureksi, voi niiden järjestelmään aiheuttama kuorma kääntyä niiden tuottamia hyötyjä vastaan. Oman vaihtoehtonsa tarjoaa Reddit, jonka käyttämä EAV-skeema (**E**ntity-**A**ttribute-**V**alue schema) mahdollistaa kyselyiden suorittamisen pääosin pääavaimen perusteella, tehden muut indeksit pääosin tarpeettomiksi.

Edellä kuvatut menetelmät mahdollistavat relaatiotietokannan suorituskyvyn kasvattamisen, mutta näyttää kuitenkin siltä, että tutkitut esimerkkitapaukset ovat turvautuneet datan varastoinnissa ennen kaikkea datan replikointiin (replication) ja datan sirpalointiin (sharding). Tutustutaan näihin tarkemmin seuraavissa aliluvuissa.

5.6 Hajautetuista järjestelmistä

5.6.1 CAP-teoreema

Hajauttamalla ja toisintamalla tietokannan dataa useammalle palvelimelle, voidaan sekä järjestelmään kohdistuvaa luku- että kirjoituskuormitusta jakaa useammalle palvelimelle. Dataa hajauttaessa ja toisintaessa on kuitenkin tarve ottaa huomioon tiettyjä hajautetuille järjestelmille ominaisia piirteitä. Gilbert ja Lynch [2002] kuvaavat Brewerin vuonna 2000 PODC-konferenssissa esittämän teoreeman mukaisesti, että hajautetun järjestelmän osalta on kolme piirrettä, joista hajautetun järjestelmän on mahdollista toteuttaa maksimissaan kaksi. Gilbertin ja Lynchin [2002] kuvaamat, CAP-kirjainlyhenteen takaa löytyvät piirteet ovat:

- *Eheys (consistency)*: järjestelmä sisältää vain yhden, ajantasaisen, version datasta.

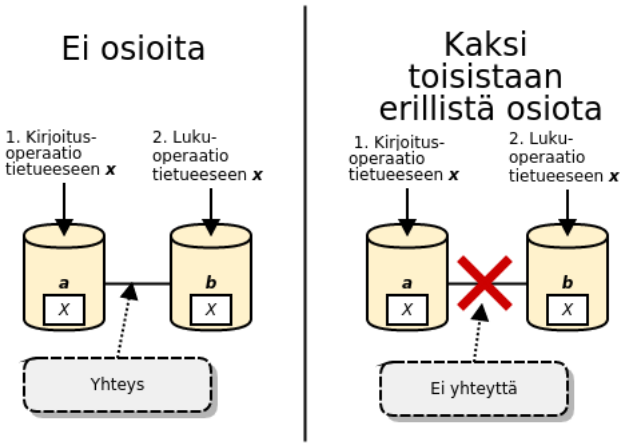
- *Saatavuus (availability)*: järjestelmän tulee kyetä vastaamaan sille lähetettyyn pyyntöön, mikäli yksikin järjestelmään kuuluva solmu on toiminnassa.

- *Osioinnin sietokyky (partition tolerance)*: kyky sietää tilanteita, joissa verkkoyhteys järjestelmän eri solmujen välillä on katkennut.

Kuva 5.3 CAP-teoreema määrittää kolme piirrettä: eheyden (C), saatavuuden (A) ja osioinnin sietokyvyn (P).

Valintojen mahdollisuutta on havainnollistettu kuvassa 5.3. Gilbertin ja Lynchin [2002] esittämää todistusta mukaillen katsotaan, miten järjestelmä toimii eri tilanteissa.

Oletetaan järjestelmä, jossa on kuvan 5.4 mukaisesti solmut a ja b. Oletetaan, että solmulle a saapuu kirjoitusoperaatio, jota seuraa samaan tietueeseen kohdistuva lukuoperaatio solmulle b. Mikäli solmu a ei ole kykenevä vastaanottamaan kirjoitusoperaatiota ilman verkkoyhteyttä solmuun b tai mikäli solmu b ei ole kykenevä vastaamaan lukuoperaatioon ilman verkkoyhteyttä solmuun a, ei järjestelmä kykene toteuttamaan osioinnin sietokykyä (P). Tilanteessa, jossa verkkoyhteys solmujen a ja b välillä vaaditaan, kykenee järjestelmä kuitenkin intuitiivisesti toteuttamaan eheys- ja saatavuuspiirteet.



Kuva 5.4 Esimerkki osioista kirjoitus- ja lukuoperaatioiden yhteydessä.

Mikäli järjestelmä on kykenevä sietämään osiointia ja solmujen a ja b välillä ei ole verkkoyhteyttä, menetetään CAP-piirteistä joko eheys tai saatavuus. Jos solmu a ei ole kykenevä ottamaan kirjoitusoperaatiota vastaan tai mikäli solmu b ei ole kykenevä vastaamaan lukuoperaatioon, voi järjestelmä kyetä pitämään sisältämänsä datan eheänä (C), mutta järjestelmä ei kykene toteuttamaan saatavuuspiirrettä (A).

Jos solmu a on kykenevä ottamaan kirjoitusoperaation vastaan, ja solmu b on kykenevä palauttamaan vastauksen lukuoperaatioon, toteuttaa järjestelmä saatavuuspiirteen (A), mutta se ei ole kykenevä toteuttamaan eheyspiirrettä, koska solmu b ei ole kykenevä palauttamaan aiemmin solmu a:lle kirjoitettua dataa.

Käytännössä osioinnin sietokyky on usein tarpeellista, ja valinta on tarpeen tehdä eheyden ja saatavuuden välillä. Muun muassa Bailis *et al.* [2013] kuvailevat eri lähteisiin viitaten, miten osioita muodostuu tietoverkkoihin yleisesti. Valinta näyttää edelleen monissa tapauksissa kallistuvan näistä kahdesta jäljelle jäävästä piirteestä saatavuuden puoleen. [Hale, 2010, Bailis & Ghodsi, 2013]

Eheys uhrataan useissa tapauksissa siten, että järjestelmä voi olla aika ajoin epäeheässä tilassa, mutta mikäli järjestelmään ei riittävän pitkään aikaan tule uusia kirjoitusoperaatiota, päätyy järjestelmä lopulta eheään tilaan. Tämänkaltaisesti toimivasta järjestelmästä käytetään termiä *lopulta yhtäpitävä* (eventually consistent). [Bailis & Ghodsi, 2013] Kaikki kolme tutkittua esimerkkitapausta käyttävät lopulta yhtäpitäviä tallennusratkaisuja.

Bailis ja Ghodsi [2013] huomauttavat lisäksi, että eheyttä uhrataan usein myös viiveiden pienentämiseksi. Mikäli kirjoitusoperaation ei tarvitse tapahtua synkronisesti kaikille järjestelmän solmuille, tai mikäli lukuoperaation ei tarvitse tarkistaa luettavan datan tuoreutta usealta solmulta, ovat operaatiot usein nopeampia suorittaa. Bailis ja Ghodsi [2013] toteavat myös, että lopulta yhtäpitävät tietokannat kykenevät usein palauttamaan itsensä eheään tilaan melko pienellä viiveellä kirjoitusoperaation jälkeen. Bailis ja Ghodsi [2013] toteavat, että esimerkiksi sarakeperhetietokanta Cassandra palauttaa itsensä monissa tilanteissa eheäksi vain 200 millisekuntia kirjoitusoperaation jälkeen.

Edberg [2013b] summaa vielä, että käytännössä järjestelmän päätyessä osioituun tilaan valinta täytyy tehdä saatavuuden ja eheyden välillä. Mikäli järjestelmä ei ole jakaantunut osioihin joudutaan valinta tekemään eheyden ja käyttäjälle näkyvän viiveen välillä.

5.6.2 BASE

ACID-piirteiden vastakohdaksi voidaan mieltää BASE-piirteet. BASE on kirjainlyhenne sanoille **B**asically **A**vailable, **S**oft state ja **E**ventually consistent. [Pritchett, 2008] Basically availablella viitataan siihen, että kaikkiin pyyntöihin kyetään vastaamaan. Vastaus ei kuitenkaan välttämättä sisällä ajantasaisinta dataa tai vastaus voi olla myös virhevastaus. Eventually consistent viittaa siihen, että mikäli järjestelmään ei muodostu järjestelmän ulkopuolelta uusia kirjoitusoperaatioita, päätyy se lopulta yhtäpitävään tilaan, jossa kaikilla järjestelmän solmuilla on käytössään ajantasaisin data. Soft state viittaa siihen, että järjestelmän tila on vähemmän vakaa kuin esimerkiksi ACID-piirteet omaavissa järjestelmissä. Järjestelmän tila voi muuttua, vaikkei käyttäjä suorittaisikaan juuri kyseisellä hetkellä kirjoitusoperaatioita siihen. Tämän piirteen aiheuttaa muun muassa se, että järjestelmä saattaa esimerkiksi itsekseen päivittää replikoitua dataa solmulta toiselle. BASE-piirteillä pyritään kuvaamaan niitä rajoitteita, joita hajautetuille järjestelmille herkästi muodostuu.

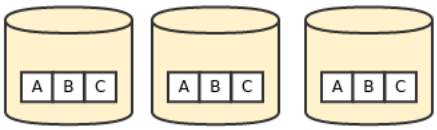
Eri piirteitä, esimerkiksi suhdetta järjestelmän eheyden ja saatavuuden välillä, on myös mahdollista vaihdella sen mukaan, mille piirteille on tarvetta. Esimerkiksi Xie *et al.* [2014] ehdottavat Salt-nimistä järjestelmää, jossa osa transaktioista suoritetaan ACID-piirteet täyttävästi ja osa ainoastaan BASE-piirteet täyttävästi. Xie *et al.* [2014] kuvaavat, että tämänkaltaisella lähestymistavalla on mahdollista nostaa järjestelmän suorituskykyä menettämättä kuitenkaan ACID-piirteitä niissä kohdin, joissa niille on aidosti tarvetta. Monet luvussa 5.10 kuvatut NoSQL-tietokannat ottavat vastaavantapaisen lähestymistavan tarjoten käyttäjilleen mahdollisuuden valita jopa jokaisen operaation kohdalla erikseen, minkälaiset eheysvaatimukset operaation tulee toteuttaa.

5.7 Replikointi

5.7.1 Yleistä

Replikoinnilla tarkoitetaan saman datan kopioimista useammalle palvelimelle kuvan 5.5 osoittamalla tavalla. Näin lukupyyntöön voi vastata mikä tahansa palvelin (replika), jolle data on kopioitu. Replikoinnin tavoitteina voi olla suorituskyvyn kasvattaminen tai vikasietoisuuden parantaminen, usein molemmat. Datan replikointia käyttää hyväkseen kaikki tutkitut esimerkkitapaukset. Näistä sekä Wikipedia että Reddit käyttävät replikointiin isäntä-orja-kokoonpanoa, jossa kirjoitusoperaatiot ohjataan isäntäpalvelimelle, joka edelleen välittää datan kaikille

orjapalvelimille. Lukupyynnöt hajautetaan orjapalvelimille.



Kuva 5.5 Replikoinnissa samat tietueet kopioidaan useille palvelimille.

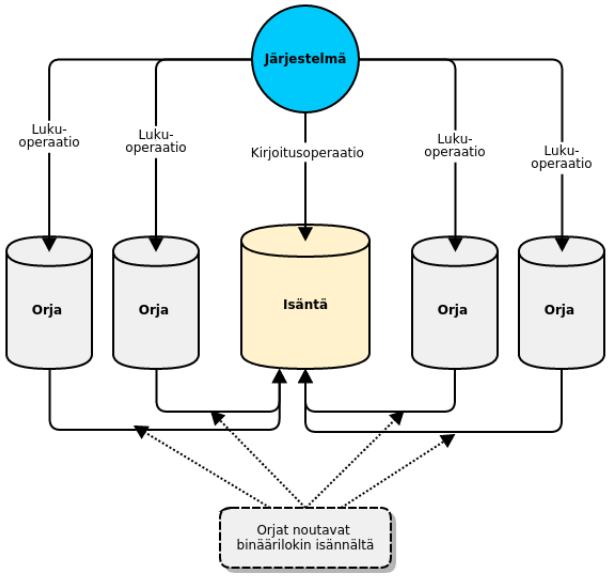
Isäntä-orja-kokoonpanoon voi kuulua yksi tai useampi isäntäpalvelin (master) sekä yksi tai useampi orjapalvelin (replikaatti, slave, replica, standby). Yksittäinen palvelin voi konfiguraatiosta riippuen toimia myös molemmissa rooleissa. Kirjoitusoperaatiot ohjataan replikoinnissa isäntäpalvelimelle ja lukuoperaatiot orjapalvelimille. Datan vastaanottamisen jälkeen data *replikoidaan* (replicate) orjapalvelimille.

Replikoinnilla voidaan keventää erityisesti yksittäiseen palvelimeen kohdistuvaa lukukuormitusta. Toisaalta myös kirjoituskuormituksen vastaanottaminen helpottuu, koska isäntäpalvelimen ei ole tarve ottaa vastaan järjestelmään kohdistuvia lukuoperaatioita.

Kuvassa 5.6 on kuvattu MySQL-relaatiotietokannan isäntä-orja-kokoonpanon tietovuota. MySQL:ssä kunkin orjan vastuulla on noutaa *binääriloki* (binary log) isännältä. MySQL:n binääriloki pitää sisällään tietokantaan tehdyt päivitykset, joten orjat kykenevät päivittämään omat tietokantansa sen pohjalta. Replikointi tapahtuu MySQL:ssä asynkronisesti ja kukin orja noutaa dataa itsellensä omaan tahtiinsa. [MySQL, 2015]

Replikointiin isännän ja orjien välillä on useita eri ratkaisumalleja. Esimerkiksi Redditin PostgreSQL:n yhteydessä käyttämä Londiste kopioi isännällä tapahtuvat muutokset PgQ-viestijonoon, jonka kautta orjat kykenevät saamaan datassa tapahtuvat muutokset itselleen.

Replikointi on mahdollista suorittaa joko synkronisesti tai asynkronisesti. Synkronisessa replikoinnissa kirjoituspyynnön lähettäjän on tarve odottaa, että kirjoitusoperaatio saadaan suoritetuksi kaikille replikointiin osallistuville palvelimille. Synkronisella replikoinnilla kyetään varmistamaan, että data on kaikilla palvelimilla varmasti ajantasaista. Vastapainona synkroninen replikointi aiheuttaa herkästi viiveitä, koska pyyntöä ei voida kuitata valmiiksi, ennen kuin synkronointi on kokonaan valmis. Asynkronisessa replikoinnissa kirjoituspyyntö kuitataan valmiiksi heti, kun se on saatu suoritettua sen vastaanottaneella palvelimella.



Kuva 5.6 Tietovuokaavio MySQL:n isäntä-orja-kokoonpanosta.

Data replikoidaan muille palvelimille vasta tämän jälkeen. Tutkituista esimerkkitapauksista sekä Reddit että Wikipedia käyttävät asynkronista replikointia.

Natanzon ja Bachmat [2013] toteavat myös, että synkronista replikointia käytettäessä järjestelmän suorituskyvyn määräytyy replikoinnissa mukana olevan heikoimman lenkin mukaan. Mikäli synkronointi tapahtuu maantieteellisesti toisistaan erillään olevien datakeskusten välillä, muodostuu myös maantieteellisestä etäisyydestä oma fysiikan lakien mukainen viiveensä, joka kannustaa asynkronisen replikoinnin puoleen [Bailis *et al.*, 2013].

Isäntä-orja-kokoonpanossa olevien palvelinten määrää ei voi kasvattaa loputtomiin. Dhamane *et al.* [2014] toteavat, että koska dataan muodostuvat päivitykset on tarve replikoida kaikille orjapalvelimille, muodostuu tästä lopulta kuorman kasvaessa pullonkaula, ja järjestelmän suorituskykyä ei ole mahdollista tietyn pisteen jälkeen enää kasvattaa vain orjainstanssien määrää lisäämällä.

Isäntä-orja-kokoonpanon haittapuoliin kuuluu isäntäpalvelin, josta muodostuu herkästi järjestelmän pullonkaula. Kuorman kasvaessa ratkaisukeinoksi muodostuu usein isäntäpalvelimen tehojen nostaminen. Tästä voi kuitenkin aiheutua tarve päivittää myös orjapalvelimet, jotta ne kykenevät edelleen vastaanottamaan isäntäpalvelimelle tallennettavan datan riittävän nopeasti. Toinen vaihtoehto on sallia konfiguraatioon useampi isäntäpalvelin.

Replikointi ei rajoitu välttämättä ainoastaan yksittäisten palvelinten välisek-

si datan kopioinniksi. Mikäli järjestelmä sijaitsee useammassa datakeskuksessa, voidaan myös kokonaiset datakeskukset mieltää isäntä-orja-rooleihin ja suorittaa replikointi näiden välillä.

Myös luvussa 5.4 kuvatut välimuistit voidaan mieltää yhdeksi replikoinnin esiintymäksi.

5.7.2 Replikointiviive ja sen käsittely

Replikointia suorittaessa yhdeksi haasteeksi muodostuu replikointiviive (replication lag). Replikointiviiveellä viitataan aikaan, joka kuluu ennen kuin isäntäpalvelimelle vastaanotettu data on kopioitu orjalle. Koska replikointia ei suoriteta heti, aiheutuu herkästi tilanne, jossa osalla solmuista on ajankohtaista dataa ja osalla ei.

Järjestelmään kohdistuvan kuorman kasvaessa myös replikointiviive usein kasvaa. Kuorman kasvaessa järjestelmän tulisi kyetä samanaikaisesti sekä palvelemaan suurempaa määrää luku- ja kirjoituspyyntöjä, että tämän ohella kyetä vastaamaan näistä pyynnöistä aiheutuneeseen suurempaan replikointikuormaan.

Luvussa 3 kuvatuista esimerkkitapauksista Wikipedia käyttää MySQL:n replikointitoimintoa. Wikipedian käyttämä MediaWiki-ohjelmisto pyrkii varmistamaan suorituskyvyn säilymisen tietyillä toimilla. Wikipedian järjestelmät muun muassa lopettavat yksittäisille orjainstanssille suoritettujen lukuoperaatioiden suorittamisen, mikäli orjainstanssin replikointiviive kasvaa yli 30 sekunnin kestoiseksi [Brown & Wilson, 2012, 184]. Lisäksi Wikipedia sisältää toiminnon, joka siirtää sen tilaan, jossa ainoastaan lukuoperaatiot ovat sallittuja, jos replikointiviive kasvaa yli 30 sekunnin kestoiseksi kaikkien orjainstanssien osalta [Brown & Wilson, 2012, 184].

Replikointiviive eri orjainstanssien välillä voi luonnollisesti aiheuttaa tilanteen, jossa eri käyttäjät näkevät eri versioita datasta. Käytännössä kuitenkin esimerkiksi Wikipedian kohdalla hieman vanhentuneen datan näkeminen ei ole kovin suuri ongelma. Artikkeli näyttää sitä lukevan käyttäjän silmiin kunnolliselta, vaikka siitä puuttuisikin jonkin toisen käyttäjän hetki sitten tekemä päivitys. Mikäli käyttäjä on tehnyt artikkeliin muutoksia, pyrkii MediaWiki-ohjelmisto suorittamaan muutosta seuraavat lukuoperaatiot sellaiselta orjainstanssilta, joka on jo noutanut kyseisen muutoksen itselleen. Täten mahdollistetaan reaaliaikaisten päivitysten esittäminen yksittäiselle käyttäjälle ja siten eheä käyttökokemus [Brown & Wilson, 2012, 184–185].

Vastaavia haasteita ja ratkaisutapoja on käytetty aikoinaan muun muassa

Facebookilla. Vuonna 2008 Facebookilla oli käytössään kaksi datakeskusta. Lukupyynnöt ohjattiin kummallekin keskukselle, mutta kirjoituspyynnöt näistä ainoastaan toiselle. Keskusten välillä esiintyneen replikointiviiveen kuvailtiin olleen 20 sekunnin luokkaa. Vastaavaan tapaan kuin Wikipediankin kohdalla, Facebookilla oli tarve kyetä näyttämään juuri kirjoitettu data käyttäjälle kirjoituspyynnön jälkeen. Facebook toteutti tämän tarpeen ohjaamalla kirjoitusoperaation suorittaneen käyttäjän lukupyynnöt 20 sekunnin ajan kirjoitusoperaation vastaanottaneelle datakeskukselle. [Sobel, 2008]

5.7.3 Päätösvaltaisuus

Replikoinnin yhteydessä voidaan kuvata päätösvaltaisuutta (quorum) muuttujien $\mathcal{N}$, $\mathcal{R}$ ja $\mathcal{W}$ avulla. Näiden muuttujien avulla voidaan määrittää se, millä varmuudella data saadaan kirjoitettua järjestelmään ja millä varmuudella luettu data on ajantasaista. Muun muassa Sadalage ja Fowler [2012, 57] kuvaavat näitä muuttujia seuraavasti:

- $\mathcal{N}$ (node): määrittää, kuinka monelle solmulle kukin järjestelmään tallennettava tietue kopioidaan.

- $\mathcal{R}$ (read): tietuetta järjestelmästä luettaessa, $\mathcal{R}$ määrittää kuinka monen solmun pitää palauttaa sama versio tietueesta, ennen kuin se voidaan lukea.

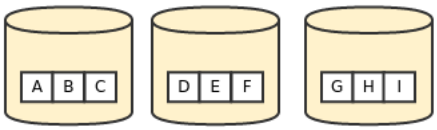
- $\mathcal{W}$ (write): tietuetta järjestelmään kirjoittaessa, $\mathcal{W}$ määrittää kuinka monen solmun tulee kuitata tietue tallennetuksi, ennen kuin tietue katsotaan onnistuneesti tallennetuksi.

Sadalage ja Fowler [2012, 57] kuvailevat, että mikäli $\mathcal{R} + \mathcal{W} > \mathcal{N}$, on luettava data aina ajantasaista. Muuttujien arvoja ja niiden välisiä suhteita muuttamalla voidaan määrittää, miten eheästi data tulee tallentaa tai lukea, ja tätä kautta vaikuttaa sekä järjestelmän piirteisiin että suorituskykyyn. Monet luvussa 5.10 kuvatut NoSQL-tietokannat sallivat $\mathcal{R}$- ja $\mathcal{W}$-asetusten määrittämisen jokaisen luku- ja kirjoitusoperaation kohdalle erikseen.

5.8 Sirpalointi

Sirpaloinnilla (sharding, horizontal partitioning) viitataan tapaan pilkkoa järjestelmään tallennettu data siten, että kullekin järjestelmän tietokantapalvelimelle

tallennetaan ainoastaan osa koko järjestelmän datasta. Tietokantapalvelimet toimivat tällaisessa tilanteessa toisistaan riippumattomasti (shared nothing). Kunkin palvelimen vastuulle jää kyseiselle palvelimelle tallennetun datan osajoukon hallinnointi. Koska sirpaloidussa järjestelmässä yksittäisen palvelimen vastuulla on pienempi määrä järjestelmään kuuluvasta datasta, kohdistuu siihen täten luonnollisesti myös pienempi määrä sekä luku- että kirjoitusoperaatioita.



Kuva 5.7 Sirpaloinnissa kukin palvelin ottaa vastuun tietystä osajoukosta dataa.

Kuvassa 5.7 on esitetty esimerkki siitä, miten eri tietueet on sijoitettu eri tietokantainstansseille. Toisena vaihtoehtona jakaa dataa solmujen kesken on pilkkoa tietueet pienempiin osiin ja sijoittaa nämä eri osat eri tietokantainstansseille (vertical partitioning).

Data voidaan jakaa eri instansseille eri menetelmin. Esimerkiksi Wikipedia on pilkkonut tietokantansa siten, että jokainen eri kieliversio on sijoitettu omaan erilliseen tietokantaansa. Reddit on käyttänyt samansuuntaista tapaa, pilkkoen datan datatyypin mukaan eri palvelimille. Twitter on puolestaan aiemmin käyttänyt itse luomaansa *Gizzard*-nimistä väliohjelmistoa (middleware), jonka avulla Twitter on kyennyt automatisoimaan datan sirpaloinnin tietokantapalvelimille [Gizzard, 2013].

5.8.1 Sirpalointimenetelmät

Tavat, joilla data voidaan sirpaloida, voidaan jakaa kahteen kategoriaan: staattiseen sirpalointiin (static sharding, fixed sharding) ja dynaamiseen sirpalointiin (dynamic sharding). Staattisessa sirpaloinnissa päätös siitä, mille instanssille data kuuluu tehdään etukäteen. Koska sirpalointi vaatii yleensä erillistä sovelluskerroksen toteutusta, ei tätä päätöstä ole usein helppo muuttaa jälkikäteen. [Bell *et al.*, 2014] Esimerkiksi Wikipedia, jossa data on jaoteltu kieliversioiden mukaan, kuuluu staattisen sirpaloinnin piiriin. Dynaamisessa sirpaloinnissa käytetään erillistä hakemistoa, jonka kautta kyetään selvittämään, miltä instanssilta haettu data löytyy [Bell *et al.*, 2014]. Twitterin aiemmin käyttämä Gizzard on esimerk-

ki dynaamisesta sirpaloinnista. Gizzard ylläpitää erillistä hakemistotaulua, jonka avulla kyselyt ohjataan oikeille tietokantapalvelimille [Gizzard, 2013].

Päätös siitä, mille instanssille data kuuluu, voidaan tehdä eri menetelmin. Dataa voidaan jakaa datan kategorian mukaan, esimerkiksi Wikipediassa kieliversion mukaan (list partitioning). Toinen vaihtoehto on jakaa dataa arvojoukon mukaan (range partitioning), käyttäen esimerkiksi datan luontipäivämäärää sen määrittämiseen, mille instanssille data kuuluu. Tästä esimerkkinä löytyy Wikipedian wikitekstien tallentamiseen käyttämä external storage -tietokanta, jossa tietyin aikavälein palvelinjoukkoon lisätään uusi palvelin ja kirjoitusoperaatiot ohjataan palvelimen lisäämisen jälkeen ainoastaan kyseiselle viimeisenä lisätylle palvelimelle. Kolmannen vaihtoehdon tarjoaa hajautusarvojen laskentaan pohjautuva menetelmä (hash partitioning), johon voidaan lisäksi yhdistää luvussa 5.4.1 kuvattu johdonmukaisten hajautusarvojen käyttäminen.

5.8.2 Esimerkkitapaus, sirpalointi Instagrammilla

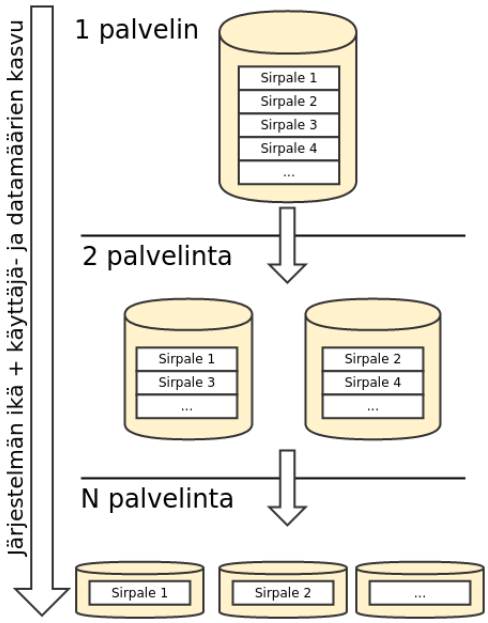
Tavallisten relaatiotietokantojen yhteydessä sirpalointia ei välttämättä ole helppoa toteuttaa jälkikäteen. Sirpalointi vaatii yleensä tukea järjestelmän sovelluskerrokselta, joten sirpaloinnin etukäteinen suunnittelu helpottaa sen toteuttamista. Yhden esimerkin tästä tarjoaa kuvien jakopalvelu Instagrammin käyttämä PostgreSQL-tietokantaympäristö. Instagrammilla sirpalointi päätettiin toteuttaa kokonaan etukäteen jakamalla tietokanta viiteentuhanteen virtuaaliseen instanssiin (logical shard, virtual shard). Virtuaaliset instanssit sijoitettiin alkuvaiheessa pienelle määrälle fyysisiä tietokantapalvelimia. Instagram loi jokaista sirpaletta varten erillisen skeeman (nimiaravuuden) PostgreSQL-tietokantaan. [Krieger, 2013] Käyttäjä- ja datamäärien kasvaessa virtuaalisia sirpaleita on ollut mahdollista siirtää suuremmalle joukolle fyysisiä palvelimia kuvan 5.8 esittämällä tavalla.

Instagrammin tarjoamasta esimerkistä voidaan tehdä kaksi johtopäätöstä:

1. Sirpalointi on mahdollista myös täysin tavallisten relaatiotietokantojen yhteydessä.

2. Sirpalointi on perinteisten relaatiotietokantojen yhteydessä helpompi toteuttaa, mikäli siihen kyetään varautumaan etukäteen.

Instagram luo pääavaimet datalle käyttäen itse kehittämäänsä algoritmia. Pääavaimen pituus on 64 bittiä ja se koostuu 41-bittisestä aikaleimasta, 13-bittisestä sirpaletunnisteesta, joka kertoo mille virtuaaliselle instanssille rivi on



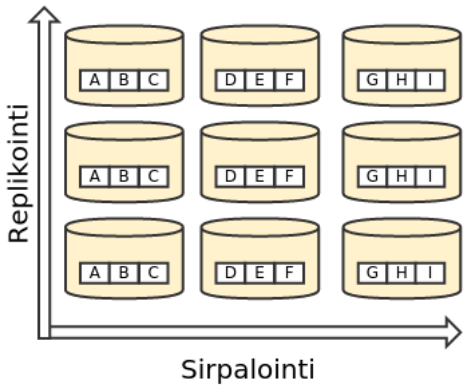
Kuva 5.8 Tietokannan jakaminen virtuaalisiin sirpaleisiin ja niiden siirtäminen useammille fyysisille palvelimille järjestelmän kuorman ja suorituskyky tarpeiden kasvaessa.

tallennettu, sekä 10-bittisestä automaattisesti kasvavasta kokonaisluvusta. [Krieger, 2013] Tapauksissa, joissa dataa noudetaan pääavaimen perusteella, on data helppoa noutaa juuri oikealta instanssilta, koska sirpaleen tunniste on upotettu pääavaimen sisään.

Sirpalointia ja replikointia voidaan käyttää myös samanaikaisesti, kuten kuva 5.9 esittää. Sirpaloinnin ohessa tehtävällä replikoinnilla voidaan kasvattaa sekä järjestelmän luotettavuutta että suorituskykyä.

5.9 Tiedon tallentamisen monimuotoisuus

Edellä kuvattujen menetelmien ohella yksi perusta tutkielmassa käsitellyille sosiaalisen median palveluille vaikuttaa olevan useiden *erityyppisten* tietokantojen luova hyödyntäminen. Kaikki kolme tutkittua palvelua käyttävät muistinvaraisia avain-arvo-säilöjä, kuten Redistä ja Memcachedia. Näiden lisäksi taustalta löytyy relaatiotietokantoja, sarakeperhetietokantoja (column family database), graafitietokantoja (graph database) sekä säilöjä mediatiedostojen tallentamiseen ja hakuindeksien ylläpitämiseen. Tämä lähestymistapa eroaa esimerkiksi perinteisistä yritysjärjestelmistä, joissa on usein totuttu näkemään yhden operatiivisen



Kuva 5.9 Sirpalointi ja replikointi yhdessä.

järjestelmän taustalla vain yksi keskitetty relaatiotietokanta.

Sadalage ja Fowler [2012, 118–119] käyttävät tämänkaltaisesta, useiden erityyppisten, tietokantojen yhdistelystä termiä *polyglot persistence*, joka voitaisiin suomentaa esimerkiksi tiedon tallentamisen monimuotoisuudeksi. Sadalage ja Fowler toteavat, että useiden eri tietokantatyyppien käyttämisellä saavutetaan etuja, koska eri säilöt ovat erikoistuneita omiin tiettyihin tehtäviinsä, ja kykenevät siten suoriutumaan niistä tehokkaammin.

Taulukkoon 5.1 on kerätty tutkielmassa käsitellyistä palveluista löytyvät eri tietokannat. Taulukko ei todennäköisesti ole kaiken kattava, mutta taulukkoa voidaan kuitenkin pitää palveluissa esiintyvän monimuotoisuuden suhteen suuntaa antavana.

5.10 NoSQL-tietokannat

Sosiaalisen median palvelujen taustalta löytyvien tietokantojen monimuotoisuuden taustalta löytyy pääosin NoSQL-tietokantoja (**N**ot **o**nly **SQL**). NoSQL-tietokannat tarjoavat relaatiotietokantojen ohelle vaihtoehtoja datan tallentamiseen eri muodoissa, sekä usein myös automatisoituja ratkaisuja datan replikointiin, sirpalointiin ja eritasoisten eheysvaatimusten toteuttamiseen. NoSQL-tietokannat voidaan jakaa karkeasti neljään kategoriaan [Sadalage & Fowler, 2012]:

- avain-arvo-tietokantoihin
- sarakeperhetietokantoihin

- dokumenttitietokantoihin
- graafitietokantoihin.

Käydään seuraavaksi nämä neljä eri tietokantatyyppiä läpi ja katsotaan, minkälaisia piirteitä ne tarjoavat, miten ne ovat käytössä tutkielman esimerkkitapauksissa ja miten ne tukevat skaalautuvuustavoitteita.

5.10.1 Avain-arvo-tietokannat

5.10.1.1 Redis

Avain-arvo-tietokannat ovat ehkä yksinkertaisimmasta päästä NoSQL-tietokantojen joukossa. Avain-arvo-tietokannoissa kukin tietue tunnistetaan yhden avainta kuvaavan arvon perusteella. Tietueen muoto, joka avaimeen voidaan liittää, vaihtelee. Avain-arvo-tietokantoihin kuuluva Redis [Redis, 2015b] sallii esimerkiksi tietueen tyypiksi merkkijonon, hajautusarvon, listan, joukon, lajitellun joukon ja monia muita tietotyyppejä. Lisäksi Redis sallii monien operaatioiden suorittamisen näille tietuille, esimerkiksi alkioiden lisäämisen listaan. [Redis, 2015a]

Redis toimii keskusmuistin varassa, joten tämä asettaa rajoitteita sille, miten paljon dataa yhdelle Redis-instanssille on mahdollista tallentaa. Redisin vahvuuk-

		Twitter	**Reddit**	**Wikipedia**
Avain-arvo-tietokannat	Memcached	x	x	x
	Redis	x	x	x
Relaatio-tietokannat	MySQL/MariaDB			x
	PostgreSQL		x	
Sarakeperhe-tietokannat	Cassandra	x	x	
	HBase	x		
Graafi-tietokannat	FlockDB	x		
Hakuindeksit	ElasticSearch			x
	Amazon CloudSearch		x	
	Twitter Earlybird	x		
Mediatiedosto-säilöt	Amazon S3		x	
	OpenStack Swift			x

Taulukko 5.1: Eri palvelujen käyttämät datasäilöt.

siin kuuluu erilaisten tietorakenteiden tallentaminen, mutta Redis ei tarjoa tukea replikoinnille tai sirpaloinnille. Redis mahdollistaa myös datan tallentamisen massamuistille. Massamuistille dataa tallennettaessa Redis sallii eri vaihtoehtoja sille, miten luotettavasti data tulee massamuistille tallentaa ja miten paljon tämä saa vaikuttaa suorituskykyyn [Redis, 2015c].

Muun muassa Twitter ja Wikipedia käyttävät Redistä järjestelmiensä taustalla.

5.10.1.2 Memcached

Esimerkkitapausten joukossa myös Memcached [Memcached, 2015] näyttää saavuttaneen suosiota. Memcached eroaa Redisistä muun muassa sen osalta, että Memcached ei tue monimutkaisia tietorakenteita tai datan tallentamista massamuistille. Sen sijaan Memcachedin ominaispiirteisiin kuuluu erityisesti yksinkertaisuus, kaikkien operaatioiden suorittaminen $\mathcal{O}(1)$-ajassa sekä kyky toimia hajautetusti.

Kaikkien Memcachediin tallennettujen tietueiden yhteyteen tallennetaan myös erääntymisaika (expiration time), jonka jälkeen kyseinen tietue voidaan poistaa välimuistista. Uutta tietuetta lisätessä Memcached poistaa pisimpään käyttämättömänä olleen tietueen (riippumatta siitä, onko se erääntynyt vai ei), mikäli välimuistille asetettu muistiraja on täyttynyt.

Jokaiselle Memcached-instanssille on mahdollista määritellä maksimimäärä muistia, jonka se saa käyttöönsä. Tietueet Memcached kykenee sijoittamaan eri instansseille käyttäen luvussa 5.4.1 kuvattuja johdonmukaisia hajautusarvoja.

5.10.1.3 Muut

Avain-arvo-tietokantojen parista löytyy myös muita mielenkiintoisia toteutuksia, kuten muun muassa Riak [Riak, 2015a] ja LinkedInin kehittämä ja käyttämä Project Voldemort [Voldemort, 2015].

Sekä Riak että Project Voldemort tukevat sekä replikointia että sirpalointia. Lisäksi kummatkin tukevat virtuaalisolmujen käyttöä (virtual node). Sekä palvelimet että data jaetaan johdonmukaisia hajautusarvoja käyttäen kehälle (ring). Toisin kuin Redis ja Memcached, sekä Riak että Project Voldemort tallentavat tietueet myös massamuistille.

Mielenkiintoisesti sekä Riak että Project Voldemort vaikuttavat päällepäin kypsemmiltä ja monipuolisemmilta avain-arvo-tietokannoilta kuin Redis ja Memcached. Silti esimerkkitapaukset ovat päätyneet näiden sijaan Redisiin ja Memcache-

diin. Yksi syy tähän on todennäköisesti se, että sekä Redis että Memcached toimivat muistinvaraisesti ja kykenevät siten todennäköisesti tarjoamaan korkeamman suorituskyvyn.

5.10.2 Sarakeperhetietokannat

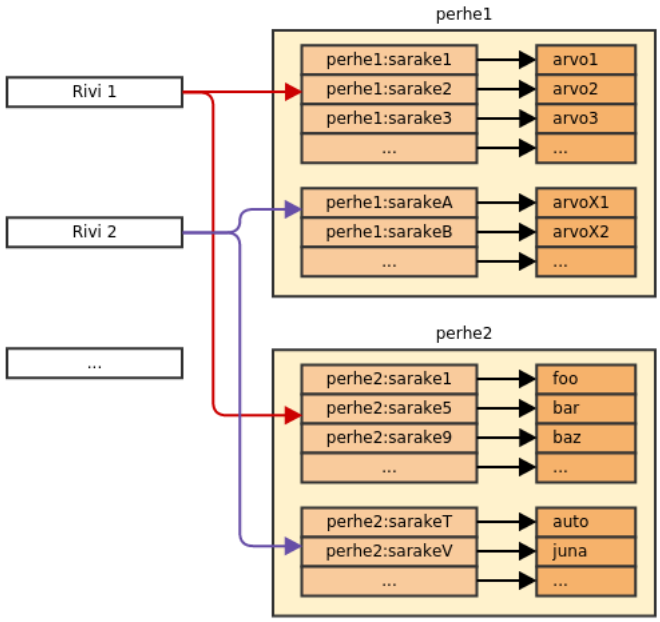
Sarakeperhetietokannat tarjoavat relaatiotietokannoista eroavan mallin tallentaa dataa. Sarakeperhetietokantojen yhteydessä puhutaan usein myös tauluista, riveistä ja sarakkeista, mutta osalla näistä termeistä on sarakeperhetietokantojen yhteydessä eri merkitys.

Sarakeperhetietokannan kantava ajatus on se, että yksi tietue muodostuu joukosta avain-arvo-pareja, jotka määritellään kullekin tietueelle erikseen. Yhteen tietueeseen voidaan liittää yhdestä aina useaan miljoonaan avain-arvo-paria. Käyttöliittymässä tietueeseen liittyvä avain-arvo-pari voidaan mieltää käyttöliittymässä näkyväksi kentäksi sekä siihen tallennetuksi arvoksi. Tämä ajattelutapa erottaa sarakeperhetietokannat esimerkiksi avain-arvo-tietokannoista, joissa yksi tietue muodostuu yhdestä avaimesta ja yhdestä siihen liitetystä arvosta. Sarakeperhetietokannassa jokaisella rivillä on uniikki tunniste, jonka avulla rivi ja siihen liittyvät avain-arvo-parit löydetään. Avaimista käytetään sarakeperhetietokantojen yhteydessä nimitystä sarake (column) ja avaimiin liitetyistä arvoista nimeä solu tai sarakkeen arvo (cell tai column value). Toisin kuin relaatiotietokannoissa, sarakeperhetietokannoissa ei tunneta tyhjää (null) arvoa. Mikäli kentässä ei ole arvoa, jätetään kyseinen avain-arvo-pari yksinkertaisesti pois rivin tiedoista.

Jokaiselle avain-arvo-parille määritellään nimiavaruus, johon se kuuluu. Tästä nimiavaruudesta käytetään termiä sarakeperhe (column family, simple column family). Sarakeperhe on tapa ryhmitellä samantapaisia avain-arvo-pareja yhdeksi joukoksi ja määritellä näille avain-arvo-pareille yhteisiä piirteitä. Yksittäiseen sarakkeeseen viitataan usein käyttämällä notaatiota *sarakeperhe:sarake*. Kuva 5.10 visualisoi sarakeperhetietokannan tietomallia.

Sarakeperheelle voi olla mahdollista määritellä esimerkiksi, että kyseiseen sarakeperheeseen kuuluvat avain-arvo-parit tulisi tiivistää (compress). Koska saman sarakeperheen alla tulisi olla samankaltaisia avain-arvo-pareja, kyetään tiivistäminen usein suorittamaan tehokkaammin kuin esimerkiksi relaatiotietokannoissa rivejä tiivistämällä. Sarakeperheelle voi olla myös mahdollista lisätä vihje siitä, että sarakeperheeseen kuuluvat sarakkeet ja niiden arvot tulisi koittaa pitää massamuistin ohella keskusmuistissa suorituskyvyn nostamiseksi.

Sarakeperhe voidaan mieltää (lajitelluksi) sanakirjaksi, jossa sarakkeiden ar-



Kuva 5.10 Visualisointi sarakeperhetietokannan tietomallista.

vot löytyvät sarakkeiden nimien avulla. Tavallisten sarakeperheiden lisäksi jotkin sarakeperhetietokannat tarjoavat ylimääräisen ulottuvuuden tiedon lajitteluun, super-sarakeperheen (super column family). Super-sarakeperhe on muuten vastaava kuin tavallinen sarakeperhe, mutta se pitää sisällään avain-arvo-sanakirjan, jossa arvot ovat sarakeperheitä. Super-sarakeperhe voidaan mieltää sanakirjaksi sarakkeita, joiden alta löytyy alisarakkeita, ja näille alisarakkeille arvoja.

Sarakeperhetietokannan tietomalli vastaa käytännössä muun muassa Redditin käyttämää EAV-skeemaa. Ei olekaan yllättävää, että Reddit on lisännyt PostgreSQL-ryppäänsä rinnalle myös sarakeperhetietokantoihin kuuluvan Cassandra-ryppään.

5.10.2.1 Bigtable

Chang *et al.* [2006] kuvasivat vuonna 2006 Googlen käyttämän Bigtable-tietokannan. Bigtable on sarakeperhetietokantoihin kuuluva, massiivisen hajautuksen mahdollistava tietokanta.

Toisin kuin relaatiotietokantaympäristöissä, Bigtablea ei ole tarkoitus suorittaa kalliilla suuren suorituskyvyn tarjoavalla palvelinkokoonpanolla. Tämän sijaan Bigtablen kantava ajatus on se, että sitä suoritetaan suurella joukolla täysin tavallisia PC-tietokoneita. Tämä johtaa vaatimuksiin kyetä hajauttamaan tieto-

kannan toiminnot suurelle joukolle palvelimia sekä kykyyn selviytyä väistämättä eteen tulevista palvelinten vikaantumisista.

Bigtablen kyky skaalautua on hyvin erilainen kuin esimerkiksi aiemmin kuvatuilla, relaatiotietokantapohjaisilla, isäntä-orja-kokoonpanoilla. Chang *et al.* [2006] kuvaavat Googlen käyttötapauksia vuodelta 2006, joissa tallennetut datamäärät vaihtelivat 0,5 teratavusta 800 teratavuun. Sarakkeisiin tallennettujen arvojen määrän vaihdellessa 0,9 miljardin ja 1000 miljardin välillä. Google kuvasi käyttäneensä tuolloin kaikkineen 24500 palvelinta Bigtablen taustalla.

Google kuvaa 8069 palvelimen ryppään kyenneen vastaamaan 1,2 miljoonaan kyselyyn sekunnissa, ottaen vastaan 741 MB/s sisäänpäin tulevaa liikennettä ja tuottaen vastineeksi 16 GB/s ulospäin lähtevää liikennettä [Chang *et al.*, 2006]. Bigtable-paperin julkaisusta on lähes vuosikymmen ja voidaan olettaa, että nykyaikaisella tekniikalla suorituskyky on vielä merkittävästi parempi.

Bigtablen kuvaillaan mahdollistavan lähes lineaarinen skaalautuvuus palvelinmäärän kasvaessa. Chang *et al.* kuvailevat, että palvelinmäärän kasvaessa 500-kertaiseksi, Bigtable-ryppään suorituskyky kyettiin kasvattamaan 300-kertaiseksi. [Chang *et al.*, 2006]

Sarakeperhepiirteiden lisäksi Bigtable mahdollistaa sarakkeisiin tallennettujen arvojen versioinnin. Versioinnin Bigtable suorittaa tallentamalla jokaisen arvon rinnalle aikaleiman. Tietty sarakkeen arvo löytyy Bigtablestä yhdistämällä rivin tunnisteen, sarakkeen nimen sekä sarakkeen arvon aikaleiman. [Chang *et al.*, 2006]

Bigtable rakentuu Googlen aiemmin rakentaman hajautetun tiedostojärjestelmän GFS:n (**G**oogle **F**ile **S**ystem) [Ghemawat *et al.*, 2003] päälle. Siinä missä Bigtable tarjoaa hajautetun tietokannan rivien ja niihin liittyvien avain-arvoparien tallentamiselle, tarjoaa GFS tiedostojärjestelmän, johon Bigtablen dataja lokitiedostot on mahdollista tallentaa luotettavasti ja hajautetusti [Chang *et al.*, 2006]. GFS tarjoaa tehokkaan tavan tallentaa, noutaa ja replikoida tiedostoja, joiden koot vaihtelevat sadoista megatavuista gigatavuihin [Ghemawat *et al.*, 2003]. Bigtable tehtäväksi muodostuu erikoistuminen näihin tiedostoihin tallennetun datan hallinnointiin.

GFS:n ohella Bigtable nojaa myös toiseen Googlen kehittämään ohjelmistoon, Chubbyyn [Burrows, 2006]. Burrows [2006] kuvailee Chubbyä hajautetuksi lukitsemispalveluksi. Chubby tarjoaa muun muassa kyvyn määrittää Bigtable-ryppäälle isäntäpalvelin. Isäntäpalvelimen määrittämisen lisäksi Bigtable hyödyntää Chubbyä metadatan tallentamiseen. Asiakasohjelman suorittaessa kyselyä Bigtable-ryppäälle, lähtee kyselyn suorittaminen liikkeelle kyselystä Chubby-ryppäälle, joka tarjoaa metadatan pohjalta asiakasohjelmalle vihjeen siitä, miltä

Bigtable-ryppään palvelimilta sen tulisi etsiä hakemaansa dataa. Chubby toimii Bigtablen tapauksessa hieman vastaavassa asemassa kuin juuripalvelimet internetin DNS-järjestelmässä (**D**omain **N**ame **S**ystem).

Bigtable itsessään ei ole käytössä tutkituissa esimerkkitapauksissa, mutta Twitterin ja Redditin käytössä olevien HBasen ja Cassandran juuret juontavat Googlen Bigtableen sekä Bigtablen taustalta löytyvään GFS:ään.

5.10.2.2 HBase

Läheisimmin Bigtablen jalanjäljissä kulkee Apache HBase -tietokanta [HBase, 2015a], joka on käytössä muun muassa Twitterillä [HBase, 2015e]. Bigtablen tavoin HBase tarjoaa merkittävän kyvyn skaalautua horisontaalisesti. HBase mainostaa itseään iskulauseella *"miljardeja rivejä x miljoonia sarakkeita"* [HBase, 2015a].

HBasen tietokanta muodostuu nimiavaruuksista, tauluista, sarakeperheistä, riveistä sekä sarakkeista. Nimiavaruuden avulla HBasessa on mahdollista ryhmitellä tauluja sekä määritellä tiettyjä piirteitä näille tauluryhmille, kuten mille palvelimille kyseisten taulujen dataa voidaan tallentaa. Rivit, sarakeperheet ja sarakkeet on edelleen ryhmitelty taulujen alle. [HBase, 2015b]

HBasessä jokainen sarake voi Bigtablen tavoin sisältää yhden tai useamman version sarakkeen arvosta. Eri versiot sarakkeen arvoista tunnistetaan arvon yhteyteen tallennettavalla aikaleimalla. Oletuksena HBase noutaa aikaleiman mukaan aina tuoreimman arvon. Sarakeperheelle on mahdollista määrittää, kuinka monta eri versiota siitä pidetään tallennettuina tai kuinka pitkään ajallisesti vanhoja versioita säilötään. Rajojen ylittyessä HBase poistaa vanhat arvot automaattisesti. [HBase, 2015d] Lisäksi HBase sallii predikaattien määrittelemisen, joiden avulla voidaan tehdä päätös vanhan arvon poistamisesta [George, 2011, 18].

HBase käsittelee yhteen riviin kohdistuvia muutoksia atomisesti. HBase ei tarjoa atomisuus- tai transaktiopiirteitä, jotka kattaisivat useampia rivejä. [HBase, 2015c] Tarvittaessa sovelluskerros voi kuitenkin hyödyntää solujen yhteyteen tallennettuja aikaleimoja monimutkaisempien eheysvaatimusten toteuttamiseen.

HBase-muodostuu ryppäästä, johon kuuluu yksi isäntäpalvelin sekä yksi tai useampi orjapalvelin. Vastaavaan tapaan kuin saraketietokannan sarake ei suoraan vastaa relaatiotietokannan saraketta, eivät myöskään termit isäntä- tai orjapalvelin vastaa suoraan relaatiotietokantamaailman vastineita. HBase-ryppäässä isäntäpalvelimen vastuulla on valvoa orjapalvelimia sekä esimerkiksi siirtää HBase-

tietokannan sirpaleita suuren kuorman alaisilta orjapalvelimilta pienemmän kuorman alaisille orjapalvelimille. Yksikään luku- tai kirjoitusoperaatio ei kulje isäntäpalvelimen kautta.

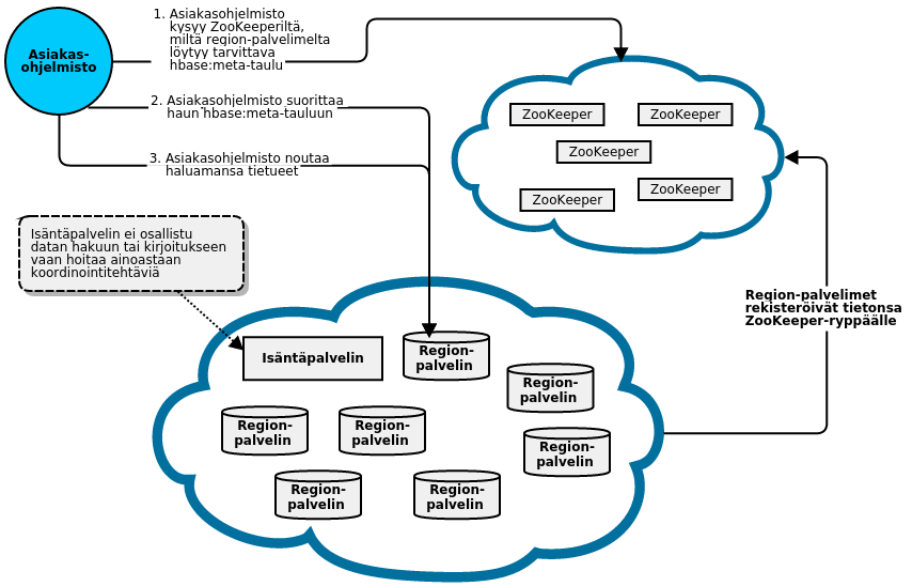
HBase tallentaa datan Region-tiedostoihin. Kunkin taulun rivit tallennetaan omiin Region-tiedostoihinsa. Rivit ovat Region-tiedostoissa järjestyksessä rivin tunnisteen mukaan. Region-tiedoston kasvaessa riittävän suureksi se laitetaan keskeltä kahtia ja sirpaloidaan kahdeksi pienemmäksi tiedostoksi. [George, 2011, 21] Koska rivit ovat järjestetty rivin tunnisteiden mukaan, käyttää HBase sirpalointimenetelmänään sirpalointia arvojoukon mukaan. Yhden Region-tiedoston kooksi suositellaan 5–20 gigatavua. [HBase, 2015b] HBase tallentaa Region-tiedostot Region-palvelimille (orjapalvelimilla). Kullekin Region-palvelimelle suositellaan tallennettavaksi 20–200 Region-tiedostoa [HBase, 2015b].

Datan jakaminen automaattisesti sopivan kokoisiin Region-tiedostoihin helpottaa kuorman jakamista eri palvelinten välillä. HBase kykenee siirtämään Region-tiedostoja tietokannan ollessa käytössä palvelimelta toiselle sen mukaan, minkälaisen kuorman alaisuudessa eri palvelimet ovat [George, 2011, 22].

HBase-rypäs vaatii rinnalleen Apache ZooKeeper-ryppään [ZooKeeper, 2015]. ZooKeeper toimii HBasen yhteydessä vastaavassa asemassa kuin Chubby Googlen Bigtablen yhteydessä. ZooKeeper ylläpitää rekisteriä siitä, mitä palvelimia HBase-ryppääseen kuuluu ja miltä palvelimilta löytyy mitäkin metadataa. Lisäksi ZooKeeperin avulla kyetään varmistamaan, että HBase-ryppäässä on ainoastaan yksi isäntäpalvelin.

Mielenkiintoiseksi kysymykseksi muodostuu se, miten HBase ylläpitää tietoa, mistä data löytyy. Dataa noutava asiakasohjelma joutuu tekemään useamman kyselyn löytääkseen etsimänsä rivit. Ensimmäinen kysely suoritetaan ZooKeeper-ryppäälle, jolta löytyy tieto siitä, miltä Region-palvelimelta sopiva *hbase:meta*-taulu löytyy. Seuraavaksi asiakasohjelma suorittaa kyselyn Region-palvelimen *hbase:meta*-tauluun selvittääkseen tarkasti, mistä asiakasohjelman etsimä data löytyy. Viimeiseksi asiakasohjelma tekee vielä erillisen kyselyn noutaakseen datan. Asiakasohjelma tallentaa omaan välimuistiinsa *hbase:meta*-taulusta haettuja tietoja, joten ensimmäisen kyselyn jälkeen seuraavat kyselyt ovat nopeampia. Kuva 5.11 esittää HBase-kokoonpanoa sekä asiakasohjelman suorittamaan kyselyä.

Skaalautuvuuspiirteiden yhtenä kustannuksena on se, että HBase on käytännössä suunniteltu ainoastaan järeään käyttöön, eli se ei sovellu pienten sovellusten tietokannaksi. Minimin HBase-kokoonpanolle kuvaillaan olevan viiden palvelimen luokkaa [HBase, 2015b]. Lisäksi HBase kaipaa rinnalleen ZooKeeper-ryppään, jonka koon tulisi olla 3–7 palvelinta [George, 2011, 62].



Kuva 5.11 HBase yleisesti.

5.10.2.3 Cassandra

Apache Cassandran [Cassandra, 2015a] kehitysjuuret juontavat Facebookiin, jonka saapuneet-kansion (inbox) hakutarpeisiin Cassandra Facebookilla alunperin kehitettiin [Lakshman, 2008].

Cassandra muistuttaa monilta piirteiltään HBasea ja sitä kautta Googlen Bigtablea. Bigtablesta ja HBasesta poiketen, Cassandra ottaa kuitenkin esimerkiksi replikointiin ja sirpalointiin mallia Amazonin DynamoDB:stä [DeCandia *et al.*, 2007]. Lisäksi toisin kuin Bigtable ja HBase, Cassandran yhteydessä ei tarvita erillistä isäntäpalvelinta, eikä se vaadi rinnalleen erillistä ZooKeeperiin tai Chubbyyn rinnastettavaa koordinoijarypästä.

Cassandra, vastaavasti kuin DynamoDB:kin, käyttää luvussa 5.4.1 kuvattuja johdonmukaisia hajautusarvoja datan sirpalointiin. Cassandra asettaa ryppääseensä kuuluvat palvelimet kehälle (ring) siten, että jokaiselle palvelimelle valitaan kehältä useampi paikka (virtual node). [Cassandra, 2015d]

Cassandrassa rivit kuuluvat aina jonkin tietyn avainavaruuden (keyspace) alle. Cassandran avainavaruus on tietyiltä piirteiltään vastaavantapainen kuin HBasestä löytyvä taulun käsite. Varsinaista taulun käsitettä Cassandrasta ei kuitenkaan löydy, joten avainavaruus on Cassandran ainoa tapa luokitella rivit. Kullekin avainavaruudelle on mahdollista määrittää tiettyjä asetuksia, kuten se, kuinka

monelle palvelimelle avaruuteen kuuluvat rivit tulisi replikoida, miten palvelimet tulisi asetella kehälle sekä mitä sarakeperheitä kyseiseen nimiavaruuteen kuuluu [Hewitt, 2010, 46].

Replikoinnin Cassandra toteuttaa vastaavasti kuin Amazonin DynamoDB [Cassandra, 2015d]. Replikoinnissa määritellään, kuinka monelle muulle palvelimelle data pitäisi kopioida. Tämän jälkeen data kopioidaan kehällä alkuperäistä palvelinta seuraaville palvelimille. [DeCandia *et al.*, 2007] Mikäli Cassandra-palvelimia on useammissa datakeskuksissa, mahdollistaa Cassandra replikointikonfiguraation, jossa data replikoidaan aina myös vähintään yhteen toiseen datakeskukseen, mahdollistaen näin paremman virheensietokyvyn. [Cassandra, 2015d]

Cassandra-ryppään palvelimet vaihtavat tietoa toisistaan juoruamisprotokollan avulla (gossip protocol). Juoruaminen tapahtuu sekunnin välein ja siinä juoruava palvelin ottaa yhteyttä kolmeen muuhun satunnaisesti valittuun palvelimeen ja vaihtaa näiden kanssa tietoja ryppään muista palvelimista [Hewitt, 2010, 88-89]. Cassandra pyrkii olemaan aina saatavilla kirjoitusoperaatioita varten. Tämä tarkoittaa sitä, että mikäli palvelin, jolle data oikeasti kuuluisi, on vikaantunut, voidaan kirjoitusoperaatio suorittaa myös jollekin toiselle palvelimelle. Mikäli oikea palvelin palaa myöhemmin takaisin ryppääseen, pyritään tämä huomaamaan juoruamisen yhteydessä ja siirtämään data tässä vaiheessa oikealle palvelimelle (hinted handoff) [Hewitt, 2010, 93].

Lukuoperaatiota käsiteltäessä, lukupyyntö ohjautuu rivin tunnisteesta lasketun hajautusarvon mukaisesti yhdelle Cassandran kehältä löytyvälle palvelimelle. Tästä palvelimesta käytetään nimitystä koordinoija (coordinator). [Kjellman, 2014] Käyttäjä voi määritellä sekä luku- että kirjoitusoperaatioiden yhteydessä sen, kuinka monen palvelimen tulee kuitata operaatio onnistuneeksi, ennen kuin pyyntöön voidaan vastata [Cassandra, 2015b]. Lukuoperaation yhteydessä tämä tarkoittaa sitä, että koordinoija tarkistaa tarvittavan määrän replikoita, ennen kuin data palautetaan käyttäjälle.

Lukuoperaation yhteydessä koordinoija suorittaa kaikille haetun avaimen sisältäville replikapalvelimille yhteenvetokyselyn (digest query) [Cassandra, 2015c]. Yhteenvedossa replikat palauttavat ainoastaan rivin sisällön pohjalta lasketun hajautusarvon sekä rivin aikaleiman [Cassandra, 2015b]. Yhteenvetokyselyn tarkoituksena on suorittaa tehokkaasti tarkistus sen suhteen, mitä versioita datasta eri replikoilla on. Mikäli replikoilta löytyy vanhentuneita versioita tietueesta, korjataan nämä lukuoperaation yhteydessä (read repair). Mikäli lukuoperaation yhteydessä annetaan vaatimus siitä, että datasta pitää löytyä ajantasaisin versio, suoritetaan vanhentuneiden versioiden korjaaminen ennen datan palauttamis-

ta. Muussa tapauksessa korjaaminen tehdään vasta datan palauttamisen jälkeen [Cassandra, 2015e].

Koska Cassandra käyttää HBasestä poiketen johdonmukaisia hajautusarvoja, ja koska Cassandrassa ei ole keskitettyä isäntää, eroaa Cassandran kuormantasaaminen HBasestä. HBasessä kuormantasauksesta vastaa isäntä, joka kykenee monitoroimaan Region-palvelimien kuormaa sekä siirtämään Region-tiedostoja raskaan kuorman alaisilta palvelimilta kevyemmän kuorman alla oleville palvelimille. Cassandrassa vastaavaa mekanismia ei ole, vaan Cassandran kohdalla on tarve luottaa siihen, että palvelimet on pilkottu riittävän moneen osaan kehälle ja kuorma jakaantuu siten riittävän tasaisesti. Mikäli erillistä kuormantasausta on tarve suorittaa, on se kuitenkin mahdollista erillisen työkalun avulla [Hewitt, 2010, 215-218].

5.10.3 Muut NoSQL-tietokannat

Avain-arvo- ja sarakeperhetietokantojen ohella myös graafitietokannat vaikuttavat tarjoavan oman lisänsä sosiaalisen median palvelujen taustalta löytyviin tietokantaratkaisuihin. Graafitietokannoille on ominaista kyky tallentaa objektien välisiä relaatioita tehokkaasti. Graafitietokannat sopivatkin siten hyvin esimerkiksi sosiaalisen median palveluissa esiintyvän sosiaalisen verkoston tallentamiseen.

Esimerkkitapausten osalta Twitter tarjoaa selvimmän esimerkin tarpeesta sosiaalisen verkoston tallentamiselle. Vähemmän yllättäen Twitterin taustalta löytyykin Twitterin itsensä kehittämä FlockDB-graafitietokanta [Pointer, 2010]. FlockDB soveltuu mainiosti esimerkiksi Twitterissä esiintyvien seuraaja-seurattu-suhteiden tallentamiseen. Wikipedian ja Redditin osalta palveluissa ei esiinny varsinaista sosiaalista verkostoa ja siten näyttää siltä, ettei niiden taustalta löydy myöskään graafitietokantoja.

Edellä mainittujen tietokantatyyppien lisäksi NoSQL-tietokantaperheeseen kuuluvat dokumenttitietokannat. Dokumenttitietokannoissa yksi tietue muodostuu yhdestä dokumentista. Dokumentti voi olla esimerkiksi JSON-tietue (**J**ava**S**cript **O**bject **N**otation). Dokumenttitietokannoissa dokumentti eroaa relaatiotietokantojen relaatioista siten, että dokumentin muotoa ei ole ennalta määritelty. Sarakeperhetietokannoista dokumenttitietokannan dokumentti eroaa sillä, että dokumenteissa ei ole rajattu sisäkkäisten sarakkeiden tai arvojen määrää. Dokumenttitietokannoissa esiintyy yhtäläisyyksiä relaatiotietokantoihin sen suhteen, että ne sallivat usein rikkaampien kyselyjen muodostamisen kuin esimerkiksi sarakeperhetietokannat. Toisaalta dokumenttitietokannat eivät välttämättä sovellu

vastaavalla tapaa tilanteisiin, joissa yksittäisellä rivillä on miljoonia sarakkeita. Dokumenttitietokannat vastaavat hyvin tilanteeseen, jossa esimerkiksi WWW-sivulla yksi dokumentti vastaa kaikesta yhdellä WWW-sivulla tarvittavasta informaatiosta.

Dokumenttitietokanta MongoDB lupailee vastaavankaltaista skaalautuvuutta kuin muutkin NoSQL-tietokannat, ja tarjoaa muun muassa tuen replikoinnille [MongoDB, 2015b] ja sirpaloinnille [MongoDB, 2015c]. Toinen suosittu dokumenttitietokanta, CouchDB [CouchDB, 2015], tarjoaa samankaltaisia piirteitä kuin MongoDB.

Esimerkkitapauksissa ei dokumenttitietokantoja esiinny, mutta esimerkkitapausten ulkopuolelta muun muassa Foursquare hyödyntää dokumenttitietokanta MongoDB:tä [MongoDB, 2015a].

5.10.4 Muita skaalautuvia toteutuksia

Googlen Bigtablen ja Amazonin DynamoDB:n jalanjäljissä on seurannut myös muita mielenkiintoisia, hajautettuja tietokantaratkaisuja. Näiden joukossa on muun muassa Yahoon kehittämä PNUTS [Cooper *et al.*, 2008] sekä Googlen myöhemmin kehittämät Megastore- sekä Spanner-tietokannat [Baker *et al.*, 2011, Corbett *et al.*, 2013]. Monista muista NoSQL-tietokannoista poiketen edellä mainitut tarjoavat myös muun muassa osittain relaationaalisia piirteitä. Lisäksi niiden skaalautuvuuspiirteet on suunniteltu esimerkiksi HBaseä ja Cassandraa suuremmiksi. Esimerkiksi Spannerin tavoitteisiin kuuluu kyky skaalautua miljoonille palvelimille ja satoihin datakeskuksiin. Lisäksi Spannerin tavoitteeksi kuvaillaan kyky tallentaa biljoonia tietueita. [Corbett *et al.*, 2013] Nämä tavoitteet luonnollisesti ylittävät esimerkiksi Redditin ja Wikipedian tarpeet.

5.11 Mediatiedostojen tallentaminen

Sosiaalisen median palveluissa yhtenä sosiaalisena objektina (tai sen osana) voivat olla mediatiedostot, kuten kuva-, ääni- ja videotiedostot. Näiden tiedostojen tallentaminen aiheuttaa omat haasteensa. Sekä Reddit että Wikipedia ovat ottaneet käyttöönsä erilliset järjestelmät, joiden avulla he huolehtivat näiden tiedostojen tallennuksesta.

Wikipedian taustalta löytyy OpenStack Swift -niminen [Swift, 2015c] järjestelmä mediatiedostojen tallentamiseen ja hallinnointiin. Swift hyödyntää vastaavankaltaisia menetelmiä siihen kohdistuvan kuormituksen hallitsemiseksi kuin aiemmin kuvatut NoSQL-tietokannat. Swift-ryppääseen kuuluvat palvelimet jae-

taan kehälle ja Swiftiin tallennettavat mediatiedostot sirpaloidaan niille lasketuttujen hajautusarvojen pohjalta. Swiftin kehälle jaetut palvelimet jaetaan lisäksi virtuaalisolmuihin. [Swift, 2015b] Lisäksi Swift tukee replikointia. [Swift, 2015a] Swiftin lisäksi Wikipedia käyttää Swiftin edustalla Varnish-välimuistipalvelimia.

Reddit on päätynyt mediatiedostojen osalta hyödyntämään ulkoisia palveluntuottajia, käyttäen Amazonin S3-palvelua mediatiedostoista luotujen pikkutiedostojen tallentamiseen. Varsinaisia mediatiedostoja Reddit ei tallenna itse ollenkaan, vaan Redditiin on mahdollista ainoastaan lisätä linkkejä esimerkiksi kuvien jakopalvelupalvelu Imguriin ja videoiden jakopalvelu Youtubeen.

5.12 Hakuindeksien tallentaminen

Palveluun tallennettu data on hyödyllistä ainoastaan, mikäli se on mahdollista saattaa käyttäjien ulottuville. Dataan on mahdollista päästä käsiksi eri keinoin. Twitterin tarjoama seuraaja-seurattu-suhde ja sen kautta seuraajille automaattisesti välittyvät viestit ovat yksi keino. Redditin ja Wikipedian tavat luokitella dataa eri kategorioihin ja mahdollistaa datan löytäminen tällä tavoin on toinen keino. Käytännössä nämä eivät kuitenkaan näytä riittäviltä, vaan kaikki kolme esimerkkitapausta mahdollistavat myös vapaamuotoisten hakujen suorittamisen.

Kaikki kolme esimerkkitapausta käyttävät erillistä alijärjestelmää hakukyselyiden prosessointiin. Erillisen alijärjestelmän käyttäminen tarkoittaa sitä, että hakujen prosessointi ei aiheuta ylimääräistä kuormaa muulle järjestelmälle. Vastapainona hakujen suorittaminen erillisessä alijärjestelmässä tarkoittaa sitä, että järjestelmän sisältämä data tulee kyetä päivittämään myös tähän erilliseen alijärjestelmään.

Twitter hyödyntää hakuindeksien tallentamiseen itse optimoimiaan Luceneinstansseja, joista se käyttää nimeä *Earlybird*. Wikipedian taustalta löytyy Luceneen ja Solr:in pohjautuva ElasticSearch, ja Reddit on päätynyt käyttämään Amazonin pilvipalveluvalikoimasta löytyvää CloudSearch-palvelua. Hakuindeksien tallentamisessa näkyy samoja tekniikoita skaalautuvuuden suhteen kuin muunkin datan tallentamisessa. Peruspiirteinä hakuindeksit sirpaloidaan ja replikoidaan useille eri palvelimille.

Myös palvelujen ulkopuoliset tahot, kuten Google, voivat indeksoida palvelujen sisältämää dataa. Muun muassa Reddit on varautunut tähän lisäkuormaan ylläpitämällä omia erillisiä palvelimia, joita käytetään palvelemaan ainoastaan Googlelta muodostuvaa kuormaa [Huffman & Williams, 2014].

5.13 Muita tapoja vaikuttaa järjestelmän kuormitukseen

Tehdään lopuksi vielä lyhyt katsaus siihen, miten järjestelmän kuormitukseen voidaan vaikuttaa, ja miten tällä voidaan mahdollisesti vaikuttaa järjestelmän kykyyn skaalautua.

5.13.1 Vanhan datan arkistointi

Dataa voidaan jakaa eri tietovarastoihin sen mukaan, minkälaisesta datasta on kyse. Eräs tapa hajauttaa järjestelmään kohdistuvaa kuormaa on datan hajauttaminen datan iän mukaan. Vanhaa dataa saatetaan käyttää hyvin eri tavoin kuin järjestelmään syötettyä uudempaa dataa. Yksi mahdollisuus järjestelmän suorituskyvyn parantamiseen onkin vanhan datan siirtäminen joko erilliseen tietokantaan tai peräti kokonaan erilliseen järjestelmään tai arkistoon. Mikäli vanhaa dataa ei kyetä siirtämään pois varsinaisen järjestelmän harteilta, aiheuttaa tämä herkästi ajan mittaan ongelmia järjestelmän ikääntyessä ja järjestelmään tallennetun datamäärän kasvaessa.

Reddit tarjoaa hyvän esimerkin datan jakamisesta sekä uuteen että tietyn aikarajan ylittäneeseen, vanhaan dataan. Redditissä viimeisimmät ja suosituimmat viestiketjut keräävät selvästi enemmän aktiivisuutta kuin vanhemmat viestiketjut. Redditin Edberg [2013b] kuvailee, että vanhan datan osalta on helpompaa, mikäli siitä päästään joko kokonaan eroon tai mikäli se kyetään ainakin saattamaan vain-luku-tilaan. Edberg [2013b] mainitsee, että palvelun käyttäjät huomaavat datan poistamisen tai sen tilan muuttamisen vain harvoin. Reddit toimii tämän ajatuksen pohjalta, ja sallii vanhojen viestiketjujen ja kommenttien osalta ainoastaan niiden lukemisen. Uusien kommenttien tai plus- ja miinuspisteiden antamista Reddit ei salli, vähentäen näin järjestelmään kohdistuvien kirjoitusoperaatioiden määrää.

Järjestelmään tallennetun datan voidaan ajatella jakautuvan neljään kategoriaan:

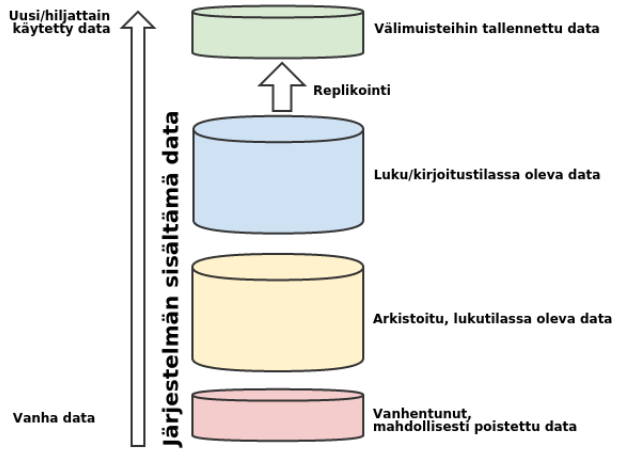
1. Välimuisteihin, jotka säilövät aktiivisessa käytössä olevan datan.

2. Järjestelmän normaaliin operatiiviseen dataan, joka on tallennettu tietokantoihin.

3. Arkistoituun dataan, joka on edelleen saavutettavissa, mutta johon ei voida enää tehdä muutoksia.

4. Dataan, joka ei syystä tai toisesta ole enää käyttäjille mielenkiintoista ja on siten poistettu järjestelmästä.

Kuvassa 5.12 havainnollistetaan tätä jakaumaa.



Kuva 5.12 Datan luokittelu datan iän mukaan.

Twitterissä datan suhteen määrittävänä tekijänä on käyttäjän aktiivisuus. Aktiivisiksi käyttäjiksi tulkitaan käyttäjät, jotka ovat käyttäneet Twitteriä viimeisen kuukauden aikana. Tällaisten käyttäjien aikajanat pidetään välimuisteissa, jotta kyseisiä käyttäjiä kyettäisiin palvelemaan nopeammin.

Edellä kuvattu datan jakautuminen ei kuitenkaan päde kaikissa tapauksissa. Esimerkiksi Wikipediassa vastaavaa vanhan tai vanhentuneen datan käsitettä ei samassa mielessä esiinny. Wikipediassa artikkelit eivät menetä ajan myötä arvoaan. Toisaalta Wikipediassakin muodostuu eroja esimerkiksi sen suhteen, kuinka suosittuja eri artikkelit ovat, ja tätä myöten tiettyjen artikkeleiden pitäminen välimuisteissa voi olla suotuisampaa kuin toisten artikkeleiden. Lisäksi Wikipedia on toteuttanut external storage -tietokantansa siten, että kirjoitusoperaatiot kohdistetaan ainoastaan ryppääseen viimeisimpänä lisätyille palvelimille. Ryppääseen lisätään ajan kuluessa uusia palvelimia. Koska ryppään vanhemmille palvelimille säilöttyyn dataan ei enää tehdä kirjoitusoperaatioita, on kyseessä käytännössä eräänlainen datan arkistointi.

Tämä malli on tuttu myös perinteisistä yritysjärjestelmistä, joissa operatiivinen data säilötään usein yhteen OLTP-tietokantaan (**On**line **T**ransaction **P**rocessing) ja tämän lisäksi data synkronoidaan erilliseen OLAP-tietokantaan (**On**line

Analytical **P**rocessing) muun muassa raporttien muodostamista varten [Fowler, 2014]. Tämänkaltaista datan replikointia on havaittavissa myös muun muassa Twitterin ja Redditin osalta. Twitter luo esimerkiksi yhteenvetosähköpostit raporttitietokanta-henkisesti [Krikorian, 2012a]. Reddit puolestaan ylläpitää eräajoin Cassandra-tietokantaryppäänsä sisältämää dataa, jota se edelleen käyttää kasvattamaan järjestelmänsä suorituskykyä [Huffman & Williams, 2014].

5.13.2 Viestijonojen käyttäminen

Järjestelmään kohdistuvan kirjoituskuormituksen käsittelyä on mahdollista siirtää myöhemmäksi viestijonojen avulla. Viestijonojen avulla järjestelmään kohdistuvia kirjoitusoperaatioita voidaan puskuroida ja puskuria voidaan tyhjentää, kun järjestelmän kuormitus on palautunut normaaliksi.

Viestijonojen käyttäminen mahdollistaa myös järjestelmän eri komponenttien välisen kommunikaation ilman suoraa komponenttien välistä yhteyttä, helpottaen täten järjestelmän komponenttien erottamista toisistaan.

5.13.3 Prosessointi eräajoina sekä prosessoinnin hajauttaminen

Yksi tapa pienentää operatiiviseen järjestelmään kohdistuvaa rasitusta on siirtää raskasta prosessointia ja analysointia myöhemmäksi, ja suorittaa sitä eräajotyylisesti. Muun muassa Wikipedia käyttää tätä tekniikka mallinteiden (template) päivitysten yhteydessä. Mallinteen päivitys on raskas operaatio, joka aiheuttaa päivityksiä kaikille mallinteen sisältäville sivuille. Wikipedia on ratkaissut tämän ongelman siirtämällä mallinteen päivitykset taustalla tapahtuviksi eräajotöiksi. [Schultz, 2014]

Lisäksi Googlen vuonna 2004 esittelemä MapReduce-laskentamalli mahdollistaa raskaiden prosessointitehtävien pilkkomiseen ja hajauttamisen suurelle määrälle palvelimia [Dean & Ghemawat, 2004]. MapReduce-laskentamalli soveltuu erityisen hyvin horisontaalisesti skaalautuvien NoSQL-tietokantojen yhteyteen. Muun muassa Reddit sekä Twitter hyödyntävät MapReduce-laskentamallia datan prosessoinnissaan [Twitter, 2012, Huffman & Williams, 2014].

5.13.4 Datan esikarsinta

Yksi tapa vähentää kuormitusta on käyttää yhdistelmää useammista eri nopeuksilla toimivista algoritmeista tehtävän suorittamiseen. Busch *et al.* [2012] antavat esimerkkinä hakupalvelut, joissa usein ensisijainen haku tehdään nopealla algoritmilla karkean hakutuloksen aikaansaamiseksi ja tämän jälkeen tähän karkeaan

hakutulokseen sovelletaan raskaampaa ja siten hitaampaa algoritmia hakutuloksen laadun parantamiseksi.

5.13.5 Rajoitusten asettaminen

Myös erilaisten rajoitusten sekä järjestelmään sisään tulevan datan sopivuuden tarkistaminen on tärkeää. Esimerkiksi Twitterin erikoispiirteisiin kuuluu viestien pituuden rajoittaminen 140 merkkiin. Vaikka rajoitteen taustalla on todennäköisesti historialliset syyt (tekstiviestien maksimipituus), auttaa se myös rajoittamaan palveluun lisättävän datan määrää.

Myös Edberg [2013b] kuvailee rajoitusten asettamisen tärkeyttä mainitsemalla tapauksen, jossa yksittäinen käyttäjä oli lisännyt Redditiin gigatavujen pituisen merkkijonon viestiketjun kommentiksi, ja aiheuttanut tällä merkittävän haitan järjestelmän suorituskyvylle.

Reddit rajoittaa muun muassa sitä, kuinka kauas ajassa taaksepäin aihealueille lisättyjä viestiketjuja voidaan selata [Huffman & Williams, 2014]. Vastaavia rajoitteita löytyy muun muassa Facebookista, jossa kaverien määrä on rajoitettu viiteentuhanteen kaveriin [Facebook, 2015b].

5.13.6 Dynaamisen sisällön muodostamisvastuun siirtäminen selaimelle

Perinteisissä WWW-sovelluksissa sivujen dynaamisesta generoinnista on vastannut järjestelmään kuulunut WWW-palvelin. WWW-palvelimen vastuulla on ollut noutaa data taustapalvelimilta ja sen jälkeen generoida HTML-sivu, johon data on ollut valmiiksi upotettu. Selaimen tehtäväksi on jäänyt ainoastaan valmiiksi generoidun HTML-sivun esittäminen.

Nykyaikaisissa yhden sivun WWW-sovelluksissa (single page application) vastuu datan upottamisesta ja siten dynaamisen HTML-sivun muodostamisesta on kuitenkin siirtynyt selaimen vastuulle. Dynaamisen HTML:n generointiin vaadittava kuorma saadaan näin siis siirrettyä pitkälti selaimen vastuulle. Yhden sivun sovelluksissa selain ylläpitää WWW-sovelluksen tilaa ja sivulta toiselle vaihdettaessa selaimen ei siten tarvitse pyytää palvelimelta kuin pieni osa niistä resursseista, jotka tarvitaan vanhemman mallisissa WWW-sovelluksissa.

Esimerkkitapauksista sekä Reddit että Wikipedia turvautuvat nykyisellään vielä HTML:n generointiin palvelinpuolella. Merkittävin syy tähän on epäilemättä palveluiden ikä ja toisaalta esimerkiksi Wikipedian kohdalla vaadittavien muutosten suuri määrä. Toisaalta esimerkiksi Wikipedia sallii pienet dynaamiset lisäykset JavaScriptiä hyödyntäen. Tällaisia lisäyksiä ovat muun muassa varojen

keruun yhteydessä käyttäjille näytettävät bannerit. Lisäksi esimerkiksi Edberg [Edberg, 2013b] toteaa Redditin osalta, että yhtenä tavoitteena on ollut muuttaa Redditin esitystapaa siten, että sivujen muodostaminen olisi kokonaan selaimen vastuulla.

Siirtämällä dynaamisen HTML:n muodostaminen selaimen vastuulle voidaan mahdollistaa myös tehokkaammat välimuistiratkaisut sekä esimerkiksi sisällönjakelun ulkoistaminen erillisille sisällönjakelupalveluille, kuten esimerkiksi Akamaille.

5.13.7 Dynaamisen sisällön vähentäminen

Dynaamisen sisällön määrää voidaan myös pyrkiä vähentämään. Muun muassa Wikipedia käyttää tätä ohjenuoraa. Tavallisella Wikipedian artikkelisivulla ei ole dynaamisia elementtejä. Tämä mahdollistaa artikkelisivujen tallentamisen sellaisenaan välimuistipalvelimille [Schultz, 2014].

Reddit hyödyntää vastaavaa tekniikkaa vanhan sisällön osalta, estäen kommenttien lisäämisen sekä plus- ja miinuspisteiden antamisen vanhoille viestiketjuille.

Lisäksi sekä Reddit että Wikipedia siirtyvät tietyissä tilanteissa tilaan, jossa ainoastaan sisällön lukeminen on mahdollista. Redditin aktiivinen sisältö tallentuu Akamain-välimuisteihin sekä Memcachediin, joten Redditin lukeminen on mahdollista, vaikka esimerkiksi taustalla olevat PostgreSQL-ryyppäät olisivat kokonaan pois käytöstä [Edberg, 2013a]. Vastaavasti Wikipedia käyttää vain-lukutilaan siirtymistä viimeisenä oljenkortenaan tilanteissa, joissa sen järjestelmiin muodostuu liiallinen kuormitus [Brown & Wilson, 2012, 184]. Wikipedia kykenee tällä tavoin helpottamaan sen isäntä-orja-kokoonpanossa olevien orjapalvelinten kuormaa ja antamaan niille tilaa kiriä kiinni replikoinnissa. Samaan aikaan suuri osa sisällöstä voidaan tarjoilla Varnish-edustapalvelimilta.

5.13.8 Virheiden salliminen

Edellä mainittujen keinojen ohella eräs tapa on sallia pienten virheiden tapahtuminen. Edberg toteaa, että järjestelmän luotettavuus on käytännössä riippuvainen siitä, miten paljon järjestelmään on käytettävissä rahaa [Edberg, 2013a]. Resursseja kasvattamalla järjestelmän luotettavuutta on mahdollista parantaa. Toisaalta sosiaalisen median kehyksessä luotettavuus ei välttämättä ole yhtä merkittävällä sijalla kuin vaikkapa Marsin pinnalle lähetettävissä avaruusluotaimissa.

Mikäli esimerkiksi Redditissä yksittäinen plus- tai miinuspiste jää tallentumatta tai Twitterissä yksittäinen twiitti katoaa, ei tätä käytännössä kukaan huomaa.

6 Loppupäätelmät

Tässä tutkielmassa on käsitelty tapoja, joilla sosiaalisen median palvelut kykenevät skaalautumaan, ja siten tallentamaan ja käsittelemään käyttäjiensä muodostamat suuret datamäärät.

Skaalautuvuuden osalta merkittäväksi piirteeksi näyttää muodostuneen kyky horisontaaliseen skaalautuvuuteen ja siten kyky pitää yksittäiseen järjestelmän resurssiin kohdistuva kuormitus kohtuullisena. Lopulta kyseessä on suurten kokonaisuuksien pilkkominen pienemmiksi ja hallittavammiksi. Vastaavaan tapaan kuin suurta ohjelmistokehitysprojektia ei ole mahdollista tehdä pilkkomatta sitä pienempiin palasiin, ei myöskään sosiaalisen median taustalta löytyviä toimintoja voi upottaa yhdelle palvelimelle tai yhteen järjestelmään. Ratkaisuksi näyttää täten muodostuneen eri toimintojen pilkkominen omiksi kokonaisuuksikseen sekä edelleen näiden kokonaisuuksien pilkkominen riittävälle määrälle toisistaan erillisiä palvelimia.

Huomion arvoista on se, että skaalautuvuuteen on selvästi jouduttu etsimään uusia ratkaisumalleja. Muun muassa Twitter, Facebook ja LinkedIn ovat Googlen ja muiden johdolla aktiivisesti kehittäneet uusia tietokanta- ja muita ratkaisuja skaalautuvuuden haasteiden ratkaisemiseksi. Sosiaalisen median taustalta löytyvät yritykset näyttävätkin enemmän ohjelmistokehitystä toteuttavilta yrityksiltä kuin miltään muulta.

Skaalautuvuus saavutetaan monissa tapauksissa vaihtamalla joitain muita ominaisuuksia pois. Näin tapahtuu esimerkiksi silloin, kun tietokantaratkaisussa päädytään hylkäämään ACID-vaateet ja tyydytään heikompiin BASE-piirteisiin. Monissa tapauksissa myös datan eheydestä luovutaan, jotta järjestelmän saatavuus, viiveet ja osioiden sietokyky kyettäisiin pitämään parempina. Monissa tapauksissa tämä luo myös tarpeen ylimääräisille ratkaisuille, joiden avulla piilotetaan esimerkiksi replikointiviiveet palvelun käyttäjiltä.

Cassandran ja HBasen kaltaiset massiivisen horisontaalisen skaalautuvuuden mahdollistavat järjestelmät ovat myös muuttaneet tapaa, jolla palvelimia käytetään. Erikoistuneet viansietokykyiset palvelimet ovat osin siirtyneet syrjemmälle. Horisontaalinen skaalautuvuus ja tietokantaratkaisuihin sisäänrakennettu viansietoisuus ovat mahdollistaneet kokoonpanot, joissa pääasemassa on kuluttajamarkkinoille suunnatut PC-laitteistot.

Tässä tutkielmassa on kuvattu eri tapoja hajauttaa järjestelmään kohdistuvaa kuormitusta, siirtää prosessointia myöhemmäksi sekä tapoja pienentää järjestelmään kohdistuvaa kuormitusta. Vaikka esimerkiksi horisontaalinen skaalautu-

vuus ja siten esimerkiksi datan sirpalointi esittävät merkittävää osaa järjestelmän skaalautuvuuden näkökulmasta, ei skaalautuvuutta kuitenkaan saavuteta millään yksittäisellä keinolla. Tutkielman esimerkkitapausten perusteella skaalautuvien järjestelmien rakentaminen muodostuukin eri ratkaisumallien yhdistelmistä. Tutkielmaan on pyritty nostamaan keskeisimmät ratkaisumallit, jotka esimerkkitapausten arkkitehtuurikuvausten perusteella on hahmotettu.

Skaalautuvuus on ajankohtainen aihe, sillä yhä useampi yritys joutuu nykymaailmassa ratkomaan big datan aiheuttamia haasteita. Skaalautuvat ratkaisut tulevat epäilemättä kehittymään ja kypsymään entisestään tulevaisuudessa. Monet tekniikat, kuten hyvän horisontaalisen skaalautuvuuden tarjoavat sarakeperhetietokannat, ovat yleistyneet vasta viime vuosina. Lisäksi esimerkiksi Googlen Megastore- sekä Spanner-tietokantojen kehitys osoittaa, ettei tarve vielä entisestään skaalautuvammille ratkaisuille ole päättynyt. Tulevaisuudessa muun muassa esineiden internet (**I**nternet **o**f **T**hings) [Atzori *et al.*, 2010] tullee lisäämään tarvetta vielä kyvykkäämmille ratkaisuille datan tallentamiseen ja hallitsemiseen.

Tutkielmassa on keskitytty skaalautuvuuteen datan tallentamisen ja käsittelyn näkökulmasta. Tämä ei kuitenkaan ole ainoa näkökulma, josta skaalautuvuutta voidaan katsoa. Skaalautuvuuden taustalta löytyy myös muita piirteitä, kuten esimerkiksi tarve kyvylle monitoroida automatisoidusti järjestelmän tilaa. Tämä aiheutuu luonnollisesti siitä, että palvelinmäärän kasvaessa ei jokaisen palvelimen tilaa ole enää mahdollista seurata manuaalisesti. Vastaavia tarpeita muodostuu muun muassa palvelinten konfiguroinnin ja päivitysten hallinnan osalta. Näiden ohella järjestelmiin kohdistuu usein esimerkiksi turvallisuus-, käytettävyys- ja saatavuusvaateita.

Järjestelmän koon kasvaessa myös ohjelmistojen eri komponenttien hallinnointi ja organisointi muodostuvat monimutkaisemmiksi. Esimerkiksi Twitterin hyödyntämä SOA-arkkitehtuuri tarjoaa tähän yhden ratkaisumallin. SOA-arkkitehtuuri tarjoaa ratkaisumallin myös kehittäjätiimien määrän kasvattamiseen, mahdollistaen eri tiimeille eri vastuualueet. Muita näkökulmia skaalautuvuuden suhteen avautuu muun muassa kustannusten hallinnasta, pilvipalvelujen hyödyntämisestä ja monesta muusta seikasta, joihin tässä tutkielmassa ei oteta kantaa.

Viiteluettelo

[Akamai, 2015] Content delivery network (cdn) and cloud computing services provider akamai. `http://www.akamai.com/`, May 2015. Noudettu: 24.05.2015.

[Alexa, 2015] The top 500 sites on the web. `http://www.alexa.com/topsites/global`, May 2015. Noudettu: 04.05.2015.

[Amazon, 2015a] Amazon cloudsearch. `http://aws.amazon.com/cloudsearch/`, May 2015. Noudettu: 09.05.2015.

[Amazon, 2015b] Amazon cloudsearch product details. `http://aws.amazon.com/cloudsearch/details/`, May 2015. Noudettu: 09.05.2015.

[Amazon, 2015c] Auto scaling. `http://aws.amazon.com/autoscaling/`, May 2015. Noudettu: 09.05.2015.

[Anhøj, 2003] Jacob Anhøj. Generic design of web-based clinical databases. *Journal of Medical Internet Research*, 5(4):e27, 2003.

[Aniszczyk, 2012] Chris Aniszczyk. Caching with twemcache. `https://blog.twitter.com/2012/caching-with-twemcache`, July 2012. Noudettu: 09.05.2015.

[Ari *et al.*, 2003] Ismail Ari, Bo Hong, Ethan L. Miller, Scott A. Brandt, & Darrell D. E. Long. Managing flash crowds on the internet. In *Proceedings of the 11th modeling, analysis, and simulation of compututer and telecommunication systems*, pages 246–249, October 2003.

[Atzori *et al.*, 2010] Luigi Atzori, Antonio Iera, & Giacomo Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787 – 2805, 2010.

[Bailis & Ghodsi, 2013] Peter Bailis & Ali Ghodsi. Eventual consistency today: Limitations, extensions, and beyond. *Queue*, 11(3):20:20–20:32, March 2013.

[Bailis *et al.*, 2013] Peter Bailis, Aaron Davidson, Alan Fekete, Ali Ghodsi, Joseph M. Hellerstein, & Ion Stoica. Highly available transactions: Virtues and limitations. *Proc. VLDB Endow.*, 7(3):181–192, November 2013.

[Baker *et al.*, 2011] Jason Baker, Chris Bond, James C. Corbett, JJ Furman, Andrey Khorlin, James Larson, Jean-Michel Leon, Yawei Li, Alexander Lloyd, & Vadim Yushprakh. Megastore: Providing scalable, highly available storage for interactive services. In *5th Biennial Conference on Innovative Data Systems Research (CIDR '11)*, 2011.

[Bell *et al.*, 2014] Charles Bell, Mats Kindahl, & Lars Thalmann. *MySQL High Availability: Tools for Building Robust Data Centers*. O'Reilly Media, Inc., 2nd edition, 2014.

[Bondi, 2000] André B. Bondi. Characteristics of scalability and their impact on performance. In *Proceedings of the 2Nd International Workshop on Software and Performance*, WOSP '00, pages 195–203. ACM, 2000.

[Boyd & Ellison, 2007] Danah M. Boyd & Nicole B. Ellison. Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication*, Volume 13(Issue 1):210 – 230, October 2007.

[Bradberry & Lubow, 2013] Russell Bradberry & Eric Lubow. *Practical Cassandra*. Pearson Education, Inc., December 2013.

[Brown & Wilson, 2012] Amy Brown & Greg Wilson. *The Architecture of Open Source Applications*, volume Volume II: Structure, Scale, and a Few More Fearless Hacks. May 2012.

[Burrows, 2006] Mike Burrows. The chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, pages 335–350. USENIX Association, 2006.

[Busch *et al.*, 2012] Michael Busch, Krishna Gade, Brian Larson, Patrick Lok, Samuel Luckenbill, & Jimmy Lin. Earlybird: Real-time search at twitter. In *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering*, ICDE '12, pages 1360–1369. IEEE Computer Society, 2012.

[Cassandra, 2015a] The apache cassandra project. `http://cassandra.apache.org/`, May 2015. Noudettu: 09.05.2015.

[Cassandra, 2015b] Configuring data consistency | datastax cassandra 2.0 documentation. http://docs.datastax.com/en/cassandra/2.0/cassandra/dml/dml_config_consistency_c.html, May 2015. Noudettu: 09.05.2015.

[Cassandra, 2015c] Digestqueries - cassandra wiki. http://wiki.apache.org/cassandra/DigestQueries, May 2015. Noudettu: 09.05.2015.

[Cassandra, 2015d] Operations - cassandra wiki. https://wiki.apache.org/cassandra/Operations, May 2015. Noudettu: 09.05.2015.

[Cassandra, 2015e] Readrepair - cassandra wiki. https://wiki.apache.org/cassandra/ReadRepair, May 2015. Noudettu: 09.05.2015.

[Chang *et al.*, 2006] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, & Robert E. Gruber. Bigtable: A distributed storage system for structured data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, OSDI '06, pages 15–15. USENIX Association, 2006.

[Chankhunthod *et al.*, 1996] Anawat Chankhunthod, Peter B. Danzig, Chuck Neerdaels, Michael F. Schwartz, & Kurt J. Worrell. A hierarchical internet object cache. In *Proceedings of the 1996 Annual Conference on USENIX Annual Technical Conference*, ATEC '96, pages 153–163. USENIX Association, 1996.

[Constine, 2012] Josh Constine. How big is facebook's data? 2.5 billion pieces of content and 500+ terabytes ingested every day. http://techcrunch.com/2012/08/22/how-big-is-facebooks-data-2-5-billion-pieces-of-content-and-500-terabytes-ingested-every-day/, August 2012. Noudettu: 01.08.2015.

[Constine, 2013] Josh Constine. Facebook is done giving its precious social graph to competitors. http://techcrunch.com/2013/01/24/my-precious-social-graph/, January 2013. Noudettu: 01.08.2015.

[Cooper *et al.*, 2008] Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel

Weaver, & Ramana Yerneni. Pnuts: Yahoo!'s hosted data serving platform. *Proc. VLDB Endow.*, 1(2):1277–1288, August 2008.

[Corbett *et al.*, 2013] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, & Dale Woodford. Spanner: Google's globally distributed database. *ACM Trans. Comput. Syst.*, 31(3):8:1–8:22, August 2013.

[CouchDB, 2015] Apache couchdb. `http://couchdb.apache.org/`, May 2015. Noudettu: 09.05.2015.

[Dean & Ghemawat, 2004] Jeffrey Dean & Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10. USENIX Association, 2004.

[DeCandia *et al.*, 2007] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, & Werner Vogels. Dynamo: Amazon's highly available key-value store. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 205–220. ACM, 2007.

[Degenhardt, 2014] Brian Degenhardt. Real-time systems at twitter. `http://www.infoq.com/presentations/real-time-twitter`, August 2014. QCon London, 2014. Noudettu: 09.05.2015.

[Dhamane *et al.*, 2014] Rohit Dhamane, Marta Patiño Martínez, Valerio Vianello, & Ricardo Jiménez Peris. Performance evaluation of database replication systems. In *Proceedings of the 18th International Database Engineering & Applications Symposium*, IDEAS '14, pages 288–293. ACM, 2014.

[Dictionary, 2015] Dictionary.com, "scalability" in collins english dictionary - complete & unabridged 10th edition. `http://dictionary.reference.com/browse/scalability?s=t`, May 2015. Noudettu: 09.05.2015.

[Duggan & Smith, 2013a] Maeve Duggan & Aaron Smith. Frequency of social media use. http://www.pewinternet.org/2013/12/30/frequency-of-social-media-use/, December 2013. Noudettu: 09.05.2015.

[Duggan & Smith, 2013b] Maeve Duggan & Aaron Smith. Social media update 2013. http://www.pewinternet.org/2013/12/30/social-media-update-2013/, December 2013. Noudettu: 09.05.2015.

[Edberg, 2013a] Jeremy Edberg. Building a reliable data store. http://www.infoq.com/presentations/Reliable-Data-Store, February 2013. QCon San Francisco, 2012. Noudettu: 09.05.2015.

[Edberg, 2013b] Jeremy Edberg. Scaling reddit from 1 million to 1 billion–pitfalls and lessons. http://www.infoq.com/presentations/scaling-reddit, August 2013. RAMP, 2012. Noudettu: 09.05.2015.

[Elmasri & Navathe, 2007] Ramez Elmasri & Shamkant B. Navathe. *Fundamentals of Database Systems*. Pearson Education, Inc., 5th edition, 2007.

[Facebook, 2015a] Company info. https://newsroom.fb.com/company-info/, January 2015. Noudettu: 21.01.2015.

[Facebook, 2015b] What is the maximum number of friends that we can add on facebook? https://www.facebook.com/help/community/question/?id=567604083305019, May 2015. Noudettu: 09.05.2015.

[Fan *et al.*, 2000] Li Fan, Pei Cao, Jussara Almeida, & Andrei Z. Broder. Summary cache: A scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Netw.*, 8(3):281–293, June 2000.

[Fatcache, 2013] fatcache. https://github.com/twitter/fatcache/blob/master/README.md, February 2013. Noudettu: 09.05.2015.

[Fowler, 2002] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, November 2002.

[Fowler, 2014] Martin Fowler. Reportingdatabase. http://martinfowler.com/bliki/ReportingDatabase.html, April 2014. Noudettu: 09.05.2015.

[Gadde *et al.*, 1997] S. Gadde, M. Rabinovich, & J. Chase. Reduce, reuse, recycle: An approach to building large internet caches. In *Proceedings of the 6th Workshop on Hot Topics in Operating Systems (HotOS-VI)*, HOTOS '97, pages 93–98. IEEE Computer Society, 1997.

[George, 2011] Lars George. *HBase: The Definitive Guide*. O'Reilly Media, Inc., September 2011.

[Ghemawat *et al.*, 2003] Sanjay Ghemawat, Howard Gobioff, & Shun-Tak Leung. The google file system. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 29–43. ACM, 2003.

[Gilbert & Lynch, 2002] Seth Gilbert & Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, June 2002.

[Gizzard, 2013] Gizzard: a library for creating distributed datastores. `https://github.com/twitter/gizzard/blob/master/README.markdown`, May 2013. Noudettu: 09.05.2015.

[Goodman, 2013] Max Goodman. Browse the future of reddit: Re-introducing multireddits. `http://www.redditblog.com/2013/06/browse-future-of-reddit-re-introducing.html`, June 2013. Noudettu: 09.05.2015.

[Goodman, 2014] Max Goodman. chromakode comments on browse the future of reddit re-introducing multireddits. `www.reddit.com/r/blog/comments/1fvovq/browse_the_future_of_reddit_reintroducing/cae97qu?context=1`, 2014. Noudettu: 09.05.2015.

[Gray & Lamport, 2006] Jim Gray & Leslie Lamport. Consensus on transaction commit. *ACM Trans. Database Syst.*, 31(1):133–160, March 2006.

[Gruener, 2012] Wolfgang Gruener. Facebook estimated to be running 180,900 servers. `www.tomshardware.com/news/facebook-servers-power-wattage-network,16961.html`, August 2012. Noudettu: 01.08.2015.

[Gualtieri, 2012] Mike Gualtieri. The pragmatic definition of big data. `http://blogs.forrester.com/mike_gualtieri/12-12-05-`

the_pragmatic_definition_of_big_data, December 2012. Noudettu: 09.05.2015.

[Hale, 2010] Coda Hale. You can't sacrifice partition tolerance. http://codahale.com/you-cant-sacrifice-partition-tolerance/, October 2010. Noudettu: 09.05.2015.

[Hansmann & Niemeyer, 2014] Thomas Hansmann & Peter Niemeyer. Big data - characterizing an emerging research field using topic models. In *Proceedings of the 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT) - Volume 01*, WI-IAT '14, pages 43–51. IEEE Computer Society, 2014.

[HAProxy, 2015] Haproxy the reliable, high performance tcp/http load balancer. http://www.haproxy.org/they-use-it.html, May 2015. Noudettu: 09.05.2015.

[Harihareswara, 2013] Sumana Harihareswara. New lua templates bring faster, more flexible pages to your wiki. https://blog.wikimedia.org/2013/03/11/lua-templates-faster-more-flexible-pages/, March 2013. Noudettu: 02.01.2015.

[Harvey, 2012] Jason Harvey. https://www.reddit.com/r/sysadmin/comments/r6zfv/we_are_sysadmins_reddit_ask_us_anything/c43f0tl, 2012. Noudettu: 09.05.2015.

[Harvey, 2014a] Jason Harvey. https://www.reddit.com/r/IAmA/comments/2ibb9t/i_am_a_reddit_employee_ama/cl0mghc, 2014. Noudettu: 09.05.2015.

[Harvey, 2014b] Jason Harvey. http://www.reddit.com/r/IAmA/comments/2ibb9t/i_am_a_reddit_employee_ama/cl0t4pp, 2014. Noudettu: 09.05.2015.

[HBase, 2015a] Apache hbase. http://hbase.apache.org, May 2015. Noudettu: 09.05.2015.

[HBase, 2015b] Apache hbase reference guide, version 2.0.0-snapshot. http://hbase.apache.org/book.html, May 2015. Noudettu: 09.05.2015.

[HBase, 2015c] Hbase – apache hbase (tm) acid properties. `http://hbase.apache.org/acid-semantics.html`, May 2015. Noudettu: 09.05.2015.

[HBase, 2015d] Hbase/datamodel. `http://wiki.apache.org/hadoop/Hbase/DataModel`, May 2015. Noudettu: 09.05.2015.

[HBase, 2015e] Hbase/poweredby. `http://wiki.apache.org/hadoop/Hbase/PoweredBy`, May 2015. Noudettu: 09.05.2015.

[Hewitt, 2010] Eben Hewitt. *Cassandra: The Definitive Guide*. O'Reilly Media, Inc., November 2010.

[Higginbotham & Kern, 2013] Stacey Higginbotham & Eliza Kern. Meet facebook's new network architecture: it's a fabric. `http://gigaom.com/2013/06/19/meet-facebooks-new-network-architecture-its-a-fabric/`, `https://gigaom.com/2013/06/19/meet-facebooks-new-network-architecture-its-a-fabric/2/`, June 2013. Noudettu: 01.08.2014.

[Huffman & Williams, 2014] Steve Huffman & Neil Williams. Web development, lesson 7: Scaling up. `https://www.udacity.com/course/web-development--cs253`, October 2014. Noudettu: 09.05.2015.

[ISO/IEC/IEEE, 2011] ISO/IEC/IEEE. Systems and software engineering – architecture description. *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)*, pages 1 – 46, January 2011.

[Karger *et al.*, 1997] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, & Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 654–663. ACM, 1997.

[Karger *et al.*, 1999] David Karger, Alex Sherman, Andy Berkheimer, Bill Bogstad, Rizwan Dhanidina, Ken Iwamoto, Brian Kim, Luke Matkins, & Yoav Yerushalmi. Web caching with consistent hashing. *Comput. Netw.*, 31(11-16):1203–1213, May 1999.

[Kietzmann *et al.*, 2011] Jan H. Kietzmann, Kristopher Hermkens, Ian P. McCarthy, & Bruno S. Silvestre. Social media? get serious! understanding the

functional building blocks of social media. *Business Horizons*, Volume 54(Issue 3):241–251, May - June 2011.

[Kiss, 2014] Jemima Kiss. Facebook's 10th birthday: from college dorm to 1.23 billion users. `http://www.theguardian.com/technology/2014/feb/04/facebook-10-years-mark-zuckerberg`, February 2014. Noudettu: 09.05.2015.

[Kjellman, 2014] Michael Kjellman. Node of the rings: Fellowship of the clusters (or: Understanding how cassandra stores data). `http://planetcassandra.org/blog/node-of-the-rings-fellowship-of-the-clusters-or-understanding-how-cassandra-stores-data/`, May 2014. Noudettu: 09.05.2015.

[Kopper, 2005] Karl Kopper. *The Linux Enterprise Cluster: Build a Highly Available Cluster with Commodity Hardware and Free Software*. No Starch Press, 2005.

[Krieger, 2013] Mike Krieger. How a small team scales instagram. `http://www.infoq.com/presentations/scaling-instagram`, December 2013. QCon San Francisco, 2013. Noudettu: 09.05.2015.

[Krikorian, 2012a] Raffi Krikorian. Real-time delivery architecture at twitter. `http://www.infoq.com/presentations/Real-Time-Delivery-Twitter`, October 2012. QCon New York, 2012. Noudettu: 09.05.2015.

[Krikorian, 2012b] Raffi Krikorian. Timelines at scale. `http://www.infoq.com/presentations/Twitter-Timeline-Scalability`, April 2012. QCon San Francisco, 2012. Noudettu: 09.05.2015.

[Krikorian, 2013] Raffi Krikorian. New tweets per second record, and how! `https://blog.twitter.com/2013/new-tweets-per-second-record-and-how`, August 2013. Noudettu: 09.05.2015.

[Krikorian, 2014] Raffi Krikorian. Scaling engineering culture at twitter. `http://www.infoq.com/presentations/Twitter-Timeline-Scalability`, February 2014. QCon San Francisco, 2013. Noudettu: 09.05.2015.

[Kruchten, 1995] Philippe B. Kruchten. The 4+1 view model of architecture. *IEEE SOFTWARE*, 12:42–50, 1995.

[Lakshman, 2008] Avinash Lakshman. Cassandra – a structured storage system on a p2p network. https://www.facebook.com/notes/facebook-engineering/cassandra-a-structured-storage-system-on-a-p2p-network/24413138919, August 2008. Noudettu: 09.05.2015.

[Lamport, 2001] Leslie Lamport. Paxos made simple. *ACM SIGACT News*, 32(4):51–58, December 2001.

[Lane, 2011] Ryan Lane. Learn about wikipedia.org architecture from the wikimedia foundation. https://www.youtube.com/watch?v=6r4qHVawVNQ, https://wikitech.wikimedia.org/w/images/f/ff/Ryan_Lane_-_Wikimedia_Architecture.pdf, April 2011. Noudettu: 09.05.2015.

[Leach *et al.*, 2005] P. Leach, M. Mealling, & R. Salz. Rfc 4122: A universally unique identifier (uuid) urn namespace. http://www.ietf.org/rfc/rfc4122.txt, 2005. Noudettu: 09.05.2015.

[Liambotis, 2014] Faidon Liambotis. dotscale 2014 - faidon liambotis - the wikimedia infrastructure. https://www.youtube.com/watch?v=646mJu5f2cQ, May 2014. dotScale Paris, 2014. Noudettu: 09.05.2015.

[Lucene, 2015] Apache lucene - apache lucene core. https://lucene.apache.org/core/, May 2015. Noudettu: 24.05.2015.

[Malpani *et al.*, 1995] Radhika Malpani, Jacob Lorch, & David Berger. Making world wide web caching servers cooperate. In *Proceedings of the Fourth International World Wide Web Conference*, pages 107–117, December 1995. http://www.w3.org/Conferences/WWW4/Papers/59/.

[Martin, 2013] Erik Martin. Top posts of 2013, stats, and snoo year's resolutions. http://www.redditblog.com/2013/12/top-posts-of-2013-stats-and-snoo-years.html, December 2013. Noudettu: 09.05.2015.

[Mediawiki, 2014] Manual:job queue. http://www.mediawiki.org/w/index.php?title=Manual:Job_queue&oldid=1063972, July 2014. Noudettu: 11.02.2015.

[Mediawiki, 2015a] Manual:cache. http://www.mediawiki.org/w/index.php?title=Manual:Cache&oldid=1396584, February 2015. Noudettu: 05.02.2015.

[Mediawiki, 2015b] Manual:mediawiki architecture. http://www.mediawiki.org/w/index.php?title=Manual:MediaWiki_architecture&oldid=1346780, January 2015. Noudettu: 09.05.2015.

[Mediawiki, 2015c] Search/old. http://www.mediawiki.org/w/index.php?title=Search/Old&oldid=1395494, February 2015. Noudettu: 11.02.2015.

[Memcached, 2015] memcached - a distributed memory object caching system. http://memcached.org/about, May 2015. Noudettu: 09.05.2015.

[MongoDB, 2015a] Foursquare | mongodb. http://www.mongodb.com/customers/foursquare, May 2015. Noudettu: 09.05.2015.

[MongoDB, 2015b] Replication introduction – mongodb manual 3.0.2. http://docs.mongodb.org/manual/core/replication-introduction/, May 2015. Noudettu: 09.05.2015.

[MongoDB, 2015c] Sharding introduction – mongodb manual 3.0.2. http://docs.mongodb.org/manual/core/sharding-introduction/, May 2015. Noudettu: 09.05.2015.

[MySQL, 2015] 16.2 replication implementation. http://dev.mysql.com/doc/refman/5.0/en/replication-implementation.html, May 2015. Noudettu: 09.05.2015.

[Natanzon & Bachmat, 2013] Assaf Natanzon & Eitan Bachmat. Dynamic synchronous/asynchronous replication. *Trans. Storage*, 9(3):8:1–8:19, August 2013.

[Pointer, 2010] Robey Pointer. Introducing flockdb. https://blog.twitter.com/2010/introducing-flockdb, May 2010. Noudettu: 09.05.2015.

[PostgreSQL, 2014] Londiste tutorial (skytools 2). https://wiki.postgresql.org/index.php?title=Londiste_Tutorial_%28Skytools_2%29&oldid=22786, June 2014. Noudettu: 09.05.2015.

[Pritchett, 2008] Dan Pritchett. Base: An acid alternative. *Queue*, 6(3):48–55, May 2008.

[Reddit, 2012] Potus iama stats. http://www.redditblog.com/2012/08/potus-iama-stats.html, August 2012. Noudettu: 09.05.2015.

[Reddit, 2015a] About reddit. http://www.reddit.com/about, May 2015. Noudettu: 09.05.2015.

[Reddit, 2015b] Architecture overview - reddit wiki - github. https://github.com/reddit/reddit/wiki/Architecture-Overview/f8a24f1081f8597b87b16367960fcf578d318804, April 2015. Noudettu: 09.05.2015.

[Redis, 2015a] An introduction to redis data types and abstractions. http://redis.io/topics/data-types-intro, May 2015. Noudettu: 09.05.2015.

[Redis, 2015b] Redis. http://redis.io/, May 2015. Noudettu: 09.05.2015.

[Redis, 2015c] Redis persistence. http://redis.io/topics/persistence, May 2015. Noudettu: 09.05.2015.

[Riak, 2015a] Riak. basho.com/riak/, May 2015. Noudettu: 09.05.2015.

[Riak, 2015b] Vnodes. http://docs.basho.com/riak/latest/theory/concepts/vnodes/, May 2015. Noudettu: 09.05.2015.

[Sadalage & Fowler, 2012] Pramod J. Sadalage & Martin Fowler. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Pearson Education, Inc., August 2012.

[Schultz, 2014] Aaron Schultz. Making wikipedia fast. https://www.youtube.com/watch?v=0PqJuZ1_B6w, May 2014. Noudettu: 09.05.2015.

[Sec, 2013a] Amendment no. 4 to form s-1. http://www.sec.gov/Archives/edgar/data/1418091/000119312513424260/d564001ds1a.htm, November 2013. Noudettu: 09.05.2015.

[Sec, 2013b] Form s-1. http://www.sec.gov/Archives/edgar/data/1418091/000119312513390321/d564001ds1.htm, October 2013. Noudettu: 09.05.2015.

[Sobel, 2008] Jason Sobel. Scaling out. https://www.facebook.com/notes/facebook-engineering/scaling-out/23844338919, August 2008. Noudettu: 09.05.2015.

[Spread, 2015] The spread toolkit. www.spread.org, May 2015. Noudettu: 09.05.2015.

[Stonebraker, 1986] Michael Stonebraker. The case for shared nothing. *Database Engineering*, 9:4–9, 1986.

[Swift, 2015a] Replication – swift 2.3.1.dev27 documentation. http://docs.openstack.org/developer/swift/overview_replication.html, May 2015. Noudettu: 09.05.2015.

[Swift, 2015b] The rings – swift 2.3.1.dev27 documentation. http://docs.openstack.org/developer/swift/overview_ring.html, May 2015. Noudettu: 09.05.2015.

[Swift, 2015c] Welcome to swift's documentation! – swift 2.3.1.dev27 documentation. http://docs.openstack.org/developer/swift/, May 2015. Noudettu: 09.05.2015.

[Tarreau, 2015] Willy Tarreau. Haproxy configuration manual version 1.6. www.haproxy.org/download/1.6/doc/configuration.txt, March 2015. Noudettu: 09.05.2015.

[Twitaholic, 2015] The twitaholic.com top 100 twitterholics based on followers. http://twitaholic.com/, May 2015. Noudettu: 09.05.2015.

[Twitter, 2011] The engineering behind twitter's new search experience. https://blog.twitter.com/2011/engineering-behind-twitter%E2%80%99s-new-search-experience, May 2011. Noudettu: 09.05.2015.

[Twitter, 2012] Generating recommendations with mapreduce and scalding. https://blog.twitter.com/2012/generating-recommendations-with-mapreduce-and-scalding, March 2012. Noudettu: 09.05.2015.

[Twitter, 2015a] Firehose. https://dev.twitter.com/streaming/firehose, May 2015. Noudettu: 09.05.2015.

[Twitter, 2015b] Post statuses/filter. https://dev.twitter.com/streaming/reference/post/statuses/filter, May 2015. Noudettu: 09.05.2015.

[Twitter, 2015c] The streaming apis. https://dev.twitter.com/streaming/overview, May 2015. Noudettu: 09.05.2015.

[Twitter, 2015d] Twitter ids. https://dev.twitter.com/overview/api/twitter-ids-json-and-snowflake, May 2015. Noudettu: 09.05.2015.

[Twitterinc, 2014] Twitter reports second quarter 2014 results. https://investor.twitterinc.com/releasedetail.cfm?ReleaseID=862505, July 2014. Noudettu: 09.05.2015.

[Unicorn, 2015] Unicorn: Rack http server for fast clients and unix. http://unicorn.bogomips.org/, May 2015. Noudettu: 09.05.2015.

[Voldemort, 2015] Project voldemort a distributed database. http://www.project-voldemort.com/voldemort/, May 2015. Noudettu: 09.05.2015.

[Wicke, 2013] Gabriel Wicke. Parsoid: How wikipedia catches up with the web. http://blog.wikimedia.org/2013/03/04/parsoid-how-wikipedia-catches-up-with-the-web/, March 2013. Noudettu: 11.02.2015.

[Wikimedia, 2014] Wikimedia servers. http://meta.wikimedia.org/w/index.php?title=Wikimedia_servers&oldid=10389266, November 2014. Noudettu: 05.02.2015.

[Wikimedia, 2015a] Page views for wikipedia, all platforms, normalized. http://stats.wikimedia.org/EN/TablesPageViewsMonthlyCombined.htm, May 2015. Noudettu: 09.05.2015.

[Wikimedia, 2015b] Statistics. http://commons.wikimedia.org/wiki/Special:Statistics, February 2015. Noudettu: 03.02.2015.

[Wikimedia, 2015c] Wikimedia statistics. http://stats.wikimedia.org/#fragment-12, May 2015. Noudettu: 09.05.2015.

[Wikimedia, 2015d] Wikipedia statistics. http://stats.wikimedia.org/EN/, March 2015. Noudettu: 09.05.2015.

[Wikimedia, 2015e] Wikipedia statistics english. http://stats.wikimedia.org/EN/TablesWikipediaEN.htm, May 2015. Noudettu: 09.05.2015.

[Wikipedia, 2014a] Help:category. http://en.wikipedia.org/w/index.php?title=Help:Category&oldid=639611265, December 2014. Noudettu: 02.01.2015.

[Wikipedia, 2014b] List of wikipedias. http://en.wikipedia.org/w/index.php?title=Wikimedia_Foundation&oldid=643105683, November 2014. Noudettu: 21.1.2015.

[Wikipedia, 2014c] Wikipedia:bots. http://en.wikipedia.org/w/index.php?title=Wikipedia:Bots&oldid=635455789, November 2014. Noudettu: 02.01.2015.

[Wikipedia, 2014d] Wikipedia:bots/status. http://en.wikipedia.org/w/index.php?title=Wikipedia:Bots/Status&oldid=612799180, June 2014. Noudettu: 02.01.2015.

[Wikipedia, 2014e] Wikipedia:list of bots by number of edits. http://en.wikipedia.org/w/index.php?title=Wikipedia:List_of_bots_by_number_of_edits&oldid=612777277, June 2014. Noudettu: 09.05.2015.

[Wikipedia, 2014f] Wikipedia:namespace. http://en.wikipedia.org/w/index.php?title=Wikipedia:Namespace&oldid=636063293, November 2014. Noudettu: 02.01.2015.

[Wikipedia, 2014g] Wikipedia:purpose. http://en.wikipedia.org/w/index.php?title=Wikipedia:Purpose&oldid=635847929, November 2014. Noudettu: 09.05.2015.

[Wikipedia, 2014h] Wikipedia:statistics. http://en.wikipedia.org/w/index.php?title=Wikipedia:Statistics&oldid=635441217, November 2014. Noudettu: 26.11.2014.

[Wikipedia, 2015a] History of wikipedia. http://en.wikipedia.org/w/index.php?title=History_of_Wikipedia&oldid=643476112, January 2015. Noudettu: 09.05.2015.

[Wikipedia, 2015b] List of social networking websites. http://en.wikipedia.org/w/index.php?title=List_of_social_networking_websites&oldid=661041371, May 2015. Noudettu: 09.05.2015.

[Wikipedia, 2015c] Mediawiki api help. http://en.wikipedia.org/w/api.php, May 2015. Noudettu: 09.05.2015.

[Wikipedia, 2015d] Wikimedia foundation. http://en.wikipedia.org/w/index.php?title=Wikimedia_Foundation&oldid=643105683, January 2015. Noudettu: 21.1.2015.

[Wikitech, 2013] Multicast htcp purging. https://wikitech.wikimedia.org/w/index.php?title=Multicast_HTCP_purging&oldid=87907, November 2013. Noudettu: 11.02.2015.

[Wikitech, 2014a] Dns. https://wikitech.wikimedia.org/w/index.php?title=DNS&oldid=132976, October 2014. Noudettu: 03.02.2015.

[Wikitech, 2014b] External storage. https://wikitech.wikimedia.org/w/index.php?title=External_storage&oldid=121081, July 2014. Noudettu: 10.02.2015.

[Wikitech, 2014c] Load balancing architecture. https://wikitech.wikimedia.org/w/index.php?title=Load_balancing_architecture&oldid=96583, January 2014. Noudettu: 10.02.2015.

[Wikitech, 2014d] Lvs. https://wikitech.wikimedia.org/w/index.php?title=LVS&oldid=115083, June 2014. Noudettu: 04.02.2015.

[Wikitech, 2014e] Mathoid - wikitech. https://wikitech.wikimedia.org/w/index.php?title=Mathoid&oldid=136267, November 2014. Noudettu: 241.05.2015.

[Wikitech, 2014f] Media storage. https://wikitech.wikimedia.org/w/index.php?title=Media_storage&oldid=113045, May 2014. Noudettu: 11.02.2015.

[Wikitech, 2014g] Redis. https://wikitech.wikimedia.org/w/index.php?title=Redis&oldid=141348, January 2014. Noudettu: 05.02.2015.

[Wikitech, 2015] Parsoid - wikitech. https://wikitech.wikimedia.org/w/index.php?title=Parsoid&oldid=143284, February 2015. Noudettu: 11.02.2015.

[Williams, 2014] Neil Williams. https://www.reddit.com/r/AskReddit/comments/2p61mu/what_excuse_are_you_sick_of_hearing/cmu3afh, 2014. Noudettu: 09.05.2015.

[Xie *et al.*, 2014] Chao Xie, Chunzhi Su, Manos Kapritsos, Yang Wang, Navid Yaghmazadeh, Lorenzo Alvisi, & Prince Mahajan. Salt: Combining acid and base in a distributed database. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, pages 495–509. USENIX Association, 2014.

[ZooKeeper, 2015] Apache zookeeper - home. https://zookeeper.apache.org/, May 2015. Noudettu: 09.05.2015.